11-Nov-2024

Internship Day - 77 Report:

Understanding Shell Scripting: Definition, Purpose, and Benefits

Definition of Shell Scripting

Shell scripting is a method of automating tasks in Unix/Linux environments by writing a series of commands in a text file, which can be executed as a program. The "shell" refers to the command-line interface that allows users to interact with the operating system. Shell scripts can include various commands, control structures (such as loops and conditionals), functions, and variables, enabling users to create complex workflows and automate repetitive tasks.

Purpose of Shell Scripting

The primary purpose of shell scripting is to simplify and automate a wide range of tasks that would otherwise require manual intervention. This is particularly useful for system administrators, developers, and anyone who frequently interacts with the command line. Shell scripts can be used for:

1. **Automation:** Automating repetitive tasks like backups, file management, and system monitoring reduces the workload and minimizes the risk of human error.

2. **Task Scheduling:** Shell scripts can be scheduled to run automatically at specified times using tools like cron, allowing for regular maintenance and updates without manual effort.

3. **System Administration:** Administrators use shell scripts to manage system configurations, user accounts, and software installations, making it easier to maintain multiple systems efficiently.

4. **Batch Processing:** Shell scripts can process large volumes of data in batch mode, enabling efficient data manipulation and analysis.

5. **Environment Setup:** They can configure the environment for development or deployment, ensuring that all necessary variables and settings are in place.

6. **Integration:** Shell scripts allow for the integration of various programs and tools, facilitating complex workflows that involve multiple steps and processes.

7. **Portability:** Shell scripts are often portable across different Unix-like systems, allowing users to run them without modification on various platforms.

8. **Simplicity:** Writing shell scripts can be simpler and quicker than developing full-fledged applications, especially for straightforward tasks.

Benefits of Shell Scripting

1) **Efficiency:** Automating tasks saves time and increases productivity by allowing users to focus on more complex and critical issues.

2) **Consistency:** Scripts ensure that tasks are performed in a consistent manner, reducing the likelihood of errors that can occur with manual execution.

3) **Flexibility:** Users can customize scripts to fit their specific needs, making it a versatile tool for a variety of tasks.

Cost-Effectiveness: Shell scripting is a cost-effective solution for automating tasks without the need for expensive software tools.

Deploying a Website Live on an Apache Server

Definition of the Script File

This shell script is designed to automate the setup of an Apache web server on a Unix/Linux system. It performs tasks such as updating the system, installing necessary packages, starting the Apache service, downloading a sample website template, and cleaning up temporary files after the installation process is complete.

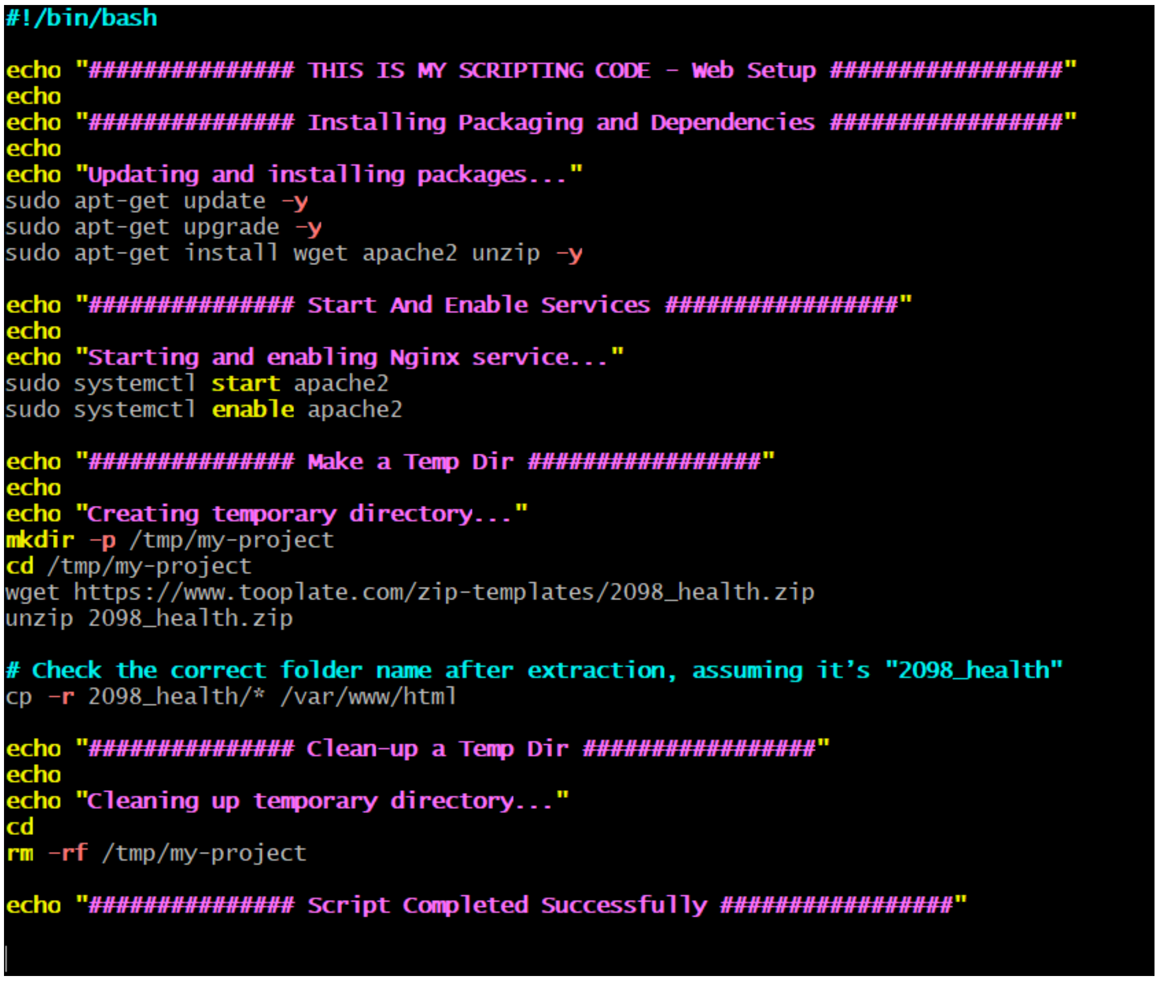
Use and Benefits:

Use: This script is used to quickly and easily set up an Apache web server and deploy a sample website. It streamlines the process of installation and configuration, making it accessible even for those with limited experience.

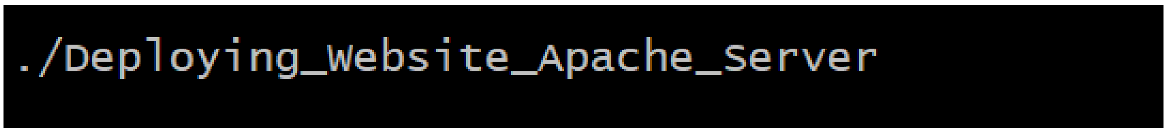
Benefits:

- Time-Saving: Automates repetitive tasks so you don't have to enter commands manually.
- Error Reduction: Reduces the chance of mistakes by executing predefined commands consistently.
- Ease of Use: Allows users to set up a web server with just a single command, making it user-friendly.
- Clean-up: Automatically removes temporary files after use, keeping the system organized.

Input:

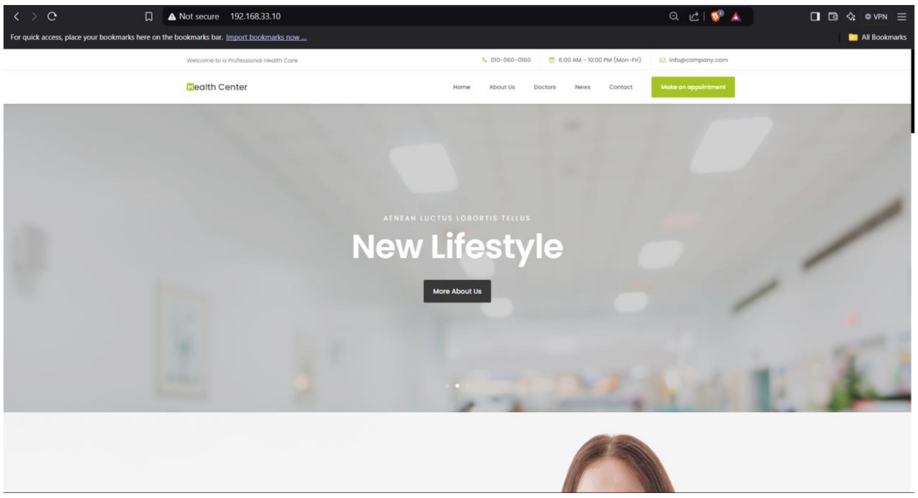


```
#!/bin/bash

echo "############### THIS IS MY SCRIPTING CODE - Web Setup #################"
echo
echo "############### Installing Packaging and Dependencies #################"
echo
echo "Updating and installing packages..."
sudo apt-get update -y
sudo apt-get upgrade -y
sudo apt-get install wget apache2 unzip -y

echo "############### Start And Enable Services #################"
echo
echo "Starting and enabling Nginx service..."
sudo systemctl start apache2
sudo systemctl enable apache2

echo "############### Make a Temp Dir #################"
echo
echo "Creating temporary directory..."
mkdir -p /tmp/my-project
cd /tmp/my-project
wget https://www.tooplate.com/zip-templates/2098_health.zip
unzip 2098_health.zip

# Check the correct folder name after extraction, assuming it's "2098_health"
cp -r 2098_health/* /var/www/html

echo "############### Clean-up a Temp Dir #################"
echo
echo "Cleaning up temporary directory..."
cd
rm -rf /tmp/my-project

echo "############### Script Completed Successfully #################"
```



```
./Deploying_Website_Apache_Server
```

Output:



Not secure 192.168.33.10

For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now ...

12-Nov-2024

Internship Day - 78 Report:

Deploying a Website Live on an NGINX Server

Description

This script automates the process of deploying a static website on a server using HTTPD (Apache). It installs necessary dependencies, downloads and unzips a website template, copies the files to the web server directory, and starts the HTTPD service to make the website live.

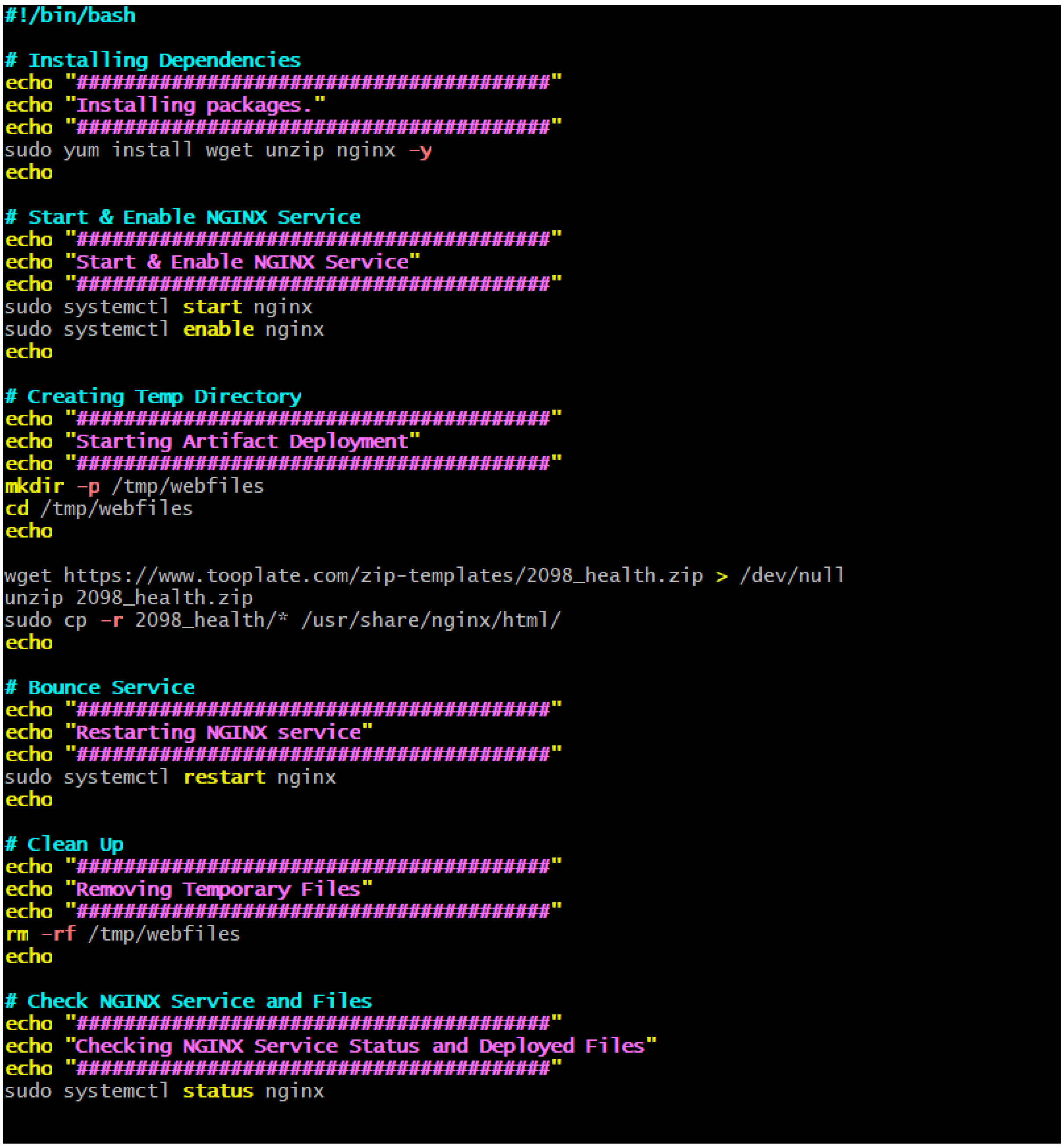
Use

This script can be used to quickly set up a static website for testing or small-scale deployment. By executing it, a user can automate the installation of dependencies, deploy web files, and ensure the HTTPD service is up and running.

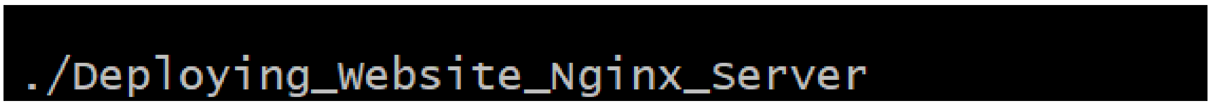
Benefits

1. **Efficiency**: Automates repetitive steps, making deployment faster.
2. **Error Reduction**: Minimizes human errors associated with manual setup.
3. **Consistency**: Ensures the same steps are followed every time, maintaining setup consistency.
4. **Convenience**: Simplifies server setup for those unfamiliar with manual deployment steps.

Input:

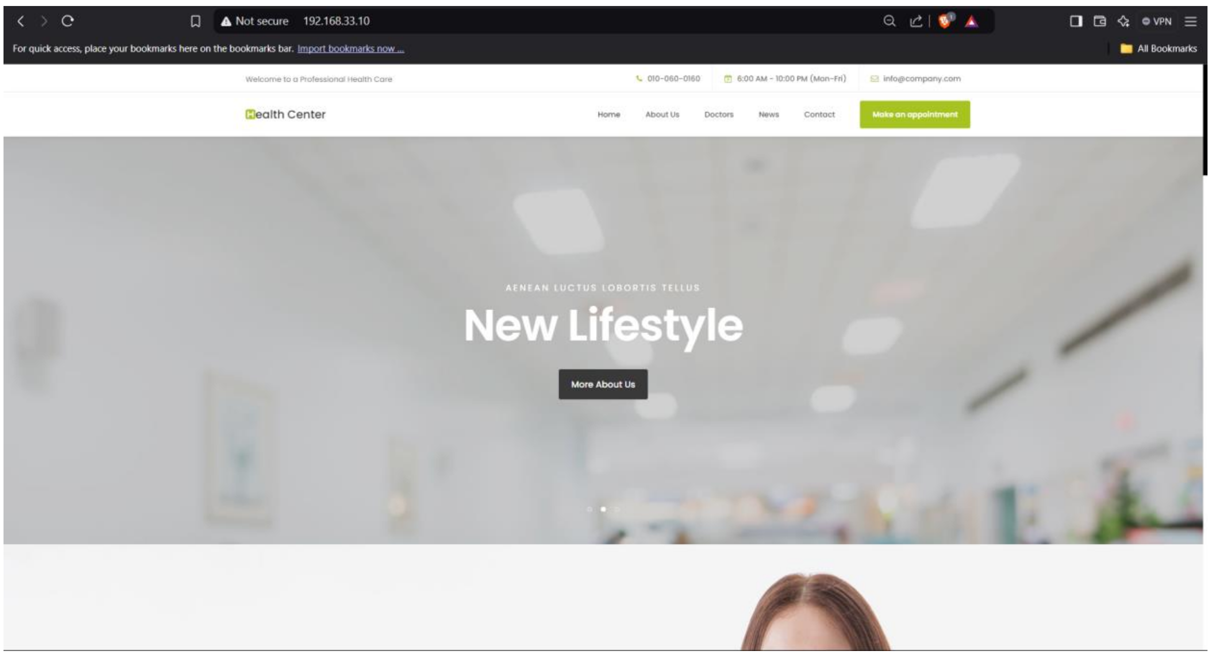


```
#!/bin/bash

# Installing Dependencies
echo "########################################"
echo "Installing packages."
echo "########################################"
sudo yum install wget unzip nginx -y
echo

# Start & Enable NGINX Service
echo "########################################"
echo "Start & Enable NGINX Service"
echo "########################################"
sudo systemctl start nginx
sudo systemctl enable nginx
echo

# Creating Temp Directory
echo "########################################"
echo "Starting Artifact Deployment"
echo "########################################"
mkdir -p /tmp/webfiles
cd /tmp/webfiles
echo

wget https://www.tooplate.com/zip-templates/2098_health.zip > /dev/null
unzip 2098_health.zip
sudo cp -r 2098_health/* /usr/share/nginx/html/
echo

# Bounce Service
echo "########################################"
echo "Restarting NGINX service"
echo "########################################"
sudo systemctl restart nginx
echo

# Clean Up
echo "########################################"
echo "Removing Temporary Files"
echo "########################################"
rm -rf /tmp/webfiles
echo

# Check NGINX Service and Files
echo "########################################"
echo "Checking NGINX Service Status and Deployed Files"
echo "########################################"
sudo systemctl status nginx
```



```
./Deploying_Website_Nginx_Server
```

Output:


Not secure 192.168.33.10
For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now ...
All Bookmarks
Welcome to a Professional Health Care
010-060-0160
6:00 AM - 10:00 PM (Mon-Fri)
info@company.com
Health Center
Home
About Us
Doctors
News
Contact
Make an appointment
AENEAN LUCTUS LOBORTIS TELLUS
New Lifestyle
More About Us

13-Nov-2024

Internship Day - 79 Report:

What is a Variable?

In shell scripting, a **variable** is a named placeholder used to store data such as strings, numbers, or command outputs. Variables make scripts more dynamic by allowing the reuse of stored values and simplifying modifications.

Key Features:

1. **No Type Declaration**: Variables in shell scripting do not require data types (e.g., int, string).
2. **Assignment**: Use = for assignment without spaces around it.
3. **Access**: Retrieve the value using a $ prefix before the variable name.

Syntax:

```
variable_name="value"  # Assign value
echo $variable_name    # Access value
```

Automated Website Deployment Script with Variable Assignments

Description

This script uses assigned variables to streamline the deployment of a static website. Variables like svc for the service name, package for required dependencies, website_url for the web template, and directory_path for the temporary storage location are defined at the beginning. This setup allows easy modification of values without needing to alter the script deeply. The script then installs necessary packages, starts and enables the HTTPD service, deploys web files from the specified URL, and cleans up temporary files after deployment.

Use

This script is useful for automating website deployment tasks on a server with minimal effort. By defining key values as variables, it allows for easy customization of the service name, package dependencies, website template URL, and directory paths. This makes it adaptable for similar deployments where these values might need changing.

Benefits

1. **Easy Customization**: Variables allow for quick updates to service names, URLs, or directories without modifying the entire script.

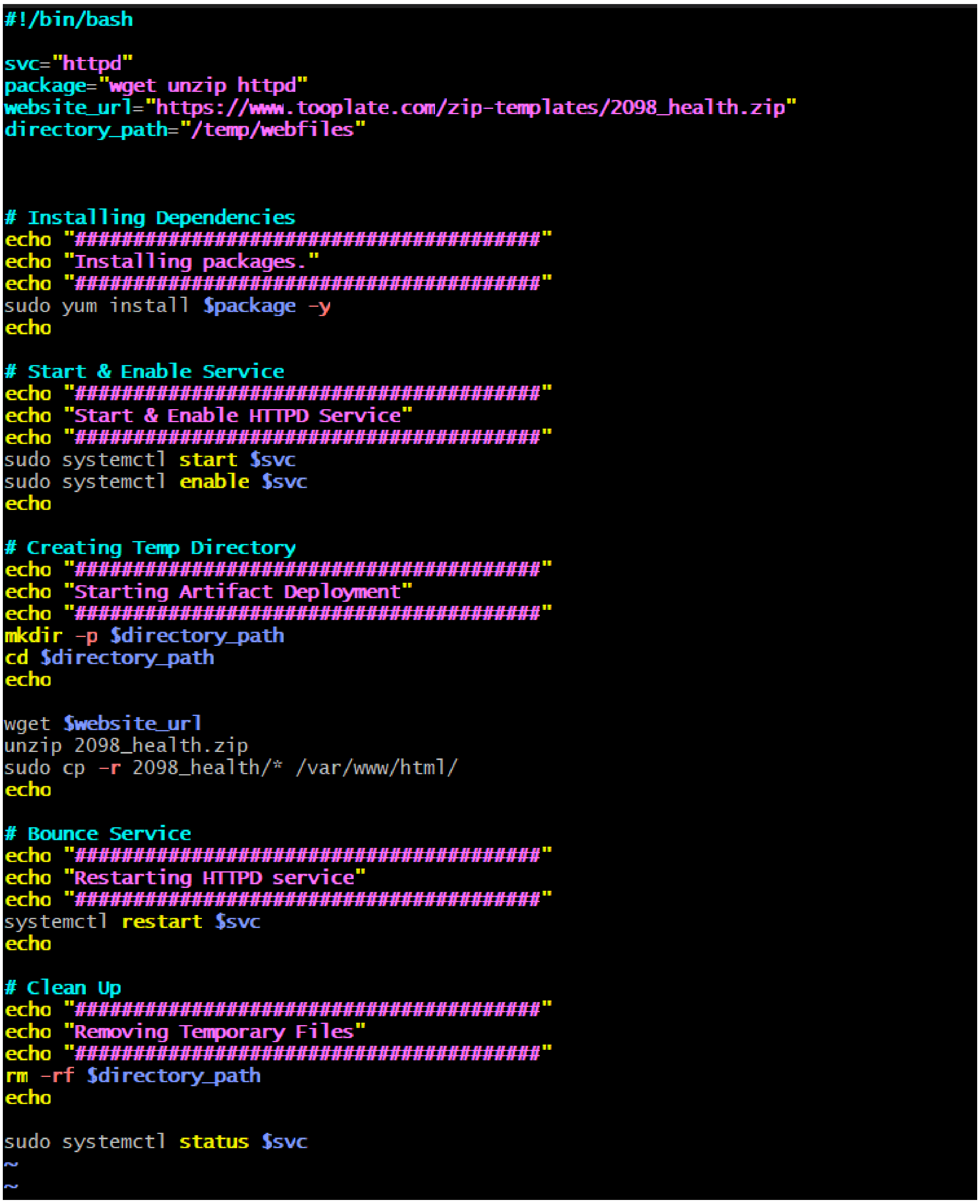
2. **Efficiency**: Automates setup steps like dependency installation and file deployment, reducing time and manual effort.

3. **Reduced Errors**: By centralizing key values in variables, the script minimizes the chance of typos or inconsistencies during configuration.

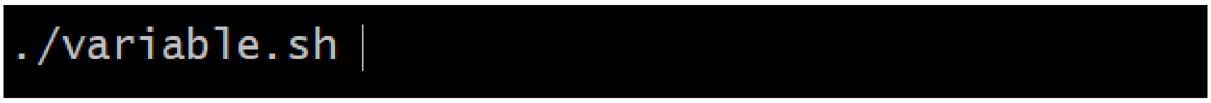
4. **Reusable**: The script can be reused across different deployments by simply adjusting variable values.

5. **Scalability**: Supports a modular approach, making it easy to add more variables if additional customization is required in future deployments.

Input:

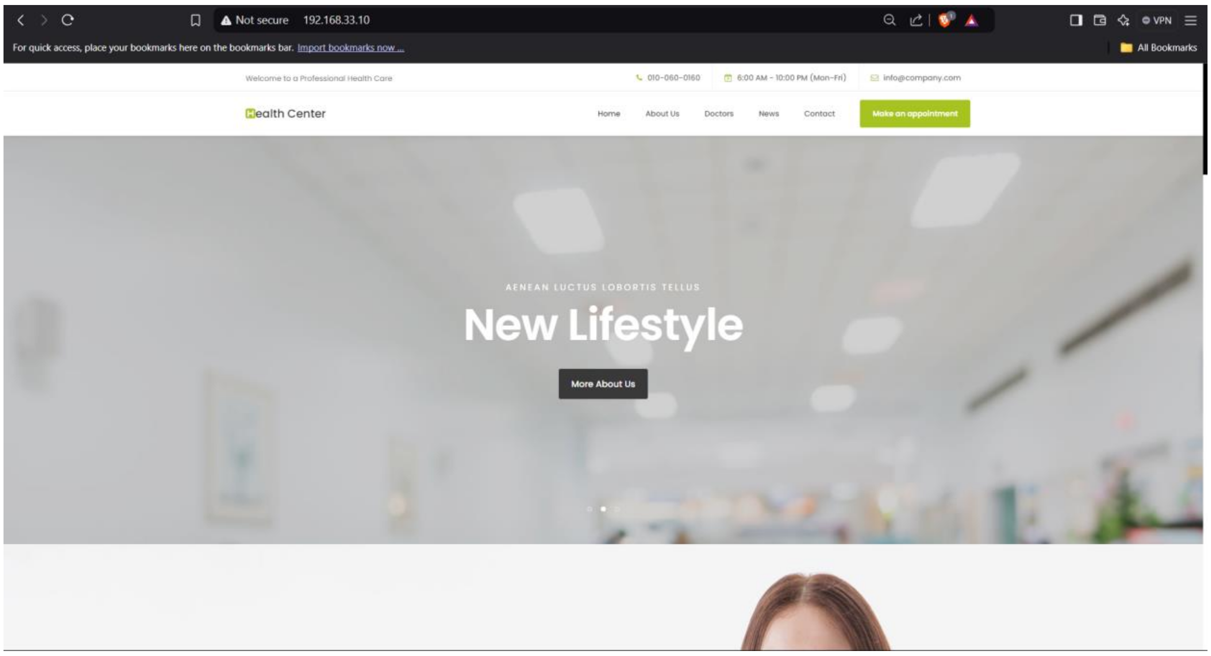


```bash
#!/bin/bash

svc="httpd"
package="wget unzip httpd"
website_url="https://www.tooplate.com/zip-templates/2098_health.zip"
directory_path="/temp/webfiles"


# Installing Dependencies
echo "########################################"
echo "Installing packages."
echo "########################################"
sudo yum install $package -y
echo

# Start & Enable Service
echo "########################################"
echo "Start & Enable HTTPD Service"
echo "########################################"
sudo systemctl start $svc
sudo systemctl enable $svc
echo

# Creating Temp Directory
echo "########################################"
echo "Starting Artifact Deployment"
echo "########################################"
mkdir -p $directory_path
cd $directory_path
echo

wget $website_url
unzip 2098_health.zip
sudo cp -r 2098_health/* /var/www/html/
echo

# Bounce Service
echo "########################################"
echo "Restarting HTTPD service"
echo "########################################"
systemctl restart $svc
echo

# Clean Up
echo "########################################"
echo "Removing Temporary Files"
echo "########################################"
rm -rf $directory_path
echo

sudo systemctl status $svc
~
~
```



```
./variable.sh
```

Output:


Not secure 192.168.33.10
For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now ...
All Bookmarks
Welcome to a Professional Health Care
010-060-0160
6:00 AM - 10:00 PM (Mon-Fri)
info@company.com
Health Center
Home
About Us
Doctors
News
Contact
Make an appointment
AENEAN LUCTUS LOBORTIS TELLUS
New Lifestyle
More About Us

Arguments ($1 $2 $3) Mean

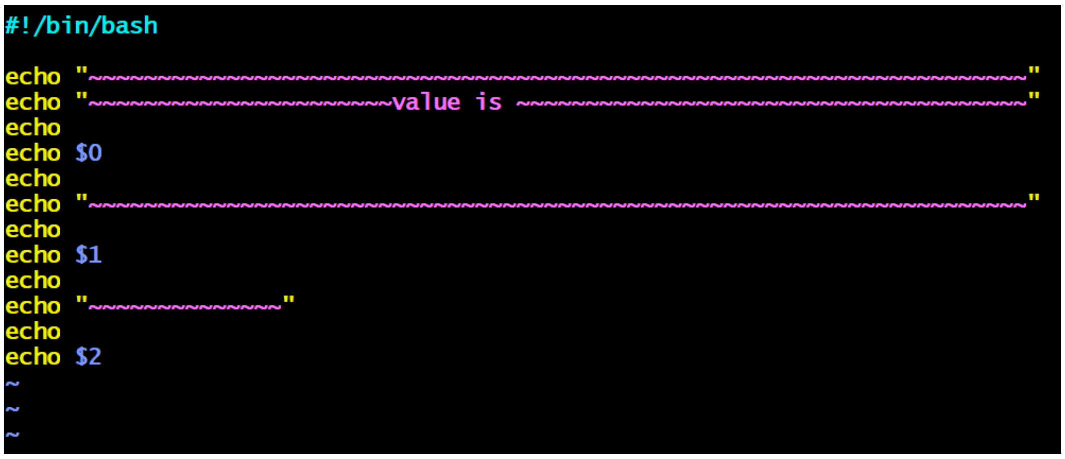
Purpose

This file is a Bash script designed to demonstrate the handling and output of positional arguments ($1, $2, etc.) in a shell script. Positional arguments allow users to pass input values to the script when it is executed.

Contents and Functionality

1. **Introduction Section**
   - Displays a visual separator ("~" characters) for formatting purposes.
   - Outputs the script name ($0) and the provided arguments ($1 and $2).
2. **Key Actions**
   - Prints the name of the script (stored in $0).
   - Prints the first ($1) and second ($2) arguments supplied when the script is executed.

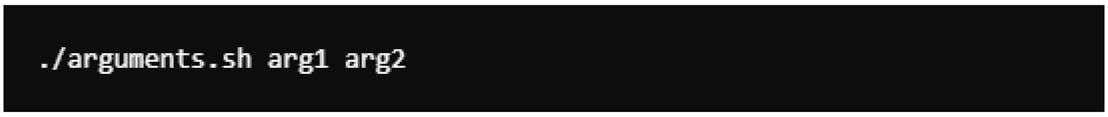
Input:



```
#!/bin/bash

echo "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
echo "~~~~~~~~~~~~~~~~~~~~~~value is ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
echo
echo $0
echo
echo "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
echo
echo $1
echo
echo "~~~~~~~~~~~~~~"
echo
echo $2
~
~
~
```

- **Command-line arguments**: The script expects two arguments to be passed when running the command.

  Example:



```
./arguments.sh arg1 arg2
```

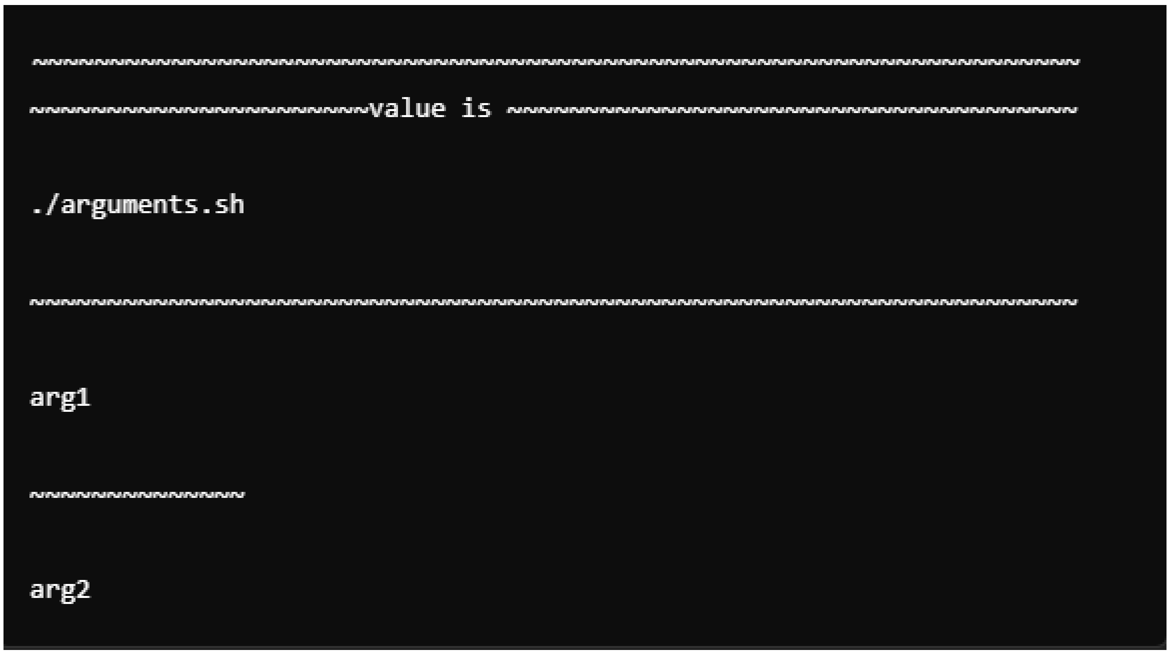
  Here, arg1 will be $1, and arg2 will be $2.

Output

The script displays:

1. The script name (e.g., ./arguments.sh).
2. The first argument ($1).
3. The second argument ($2).

Example Output:



```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~value is ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


./arguments.sh


~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


arg1


~~~~~~~~~~~~~~


arg2
```

Use Case

This script is typically used:

- To understand and debug how command-line arguments are passed to a Bash script.
- As a foundation for scripts that require user input via arguments.

Package Installation Using User Arguments

Purpose

This script demonstrates how to install software packages using yum (a package manager for RHEL-based Linux systems) based on user-provided arguments. The script allows flexibility to specify additional packages directly as input when executing the script.

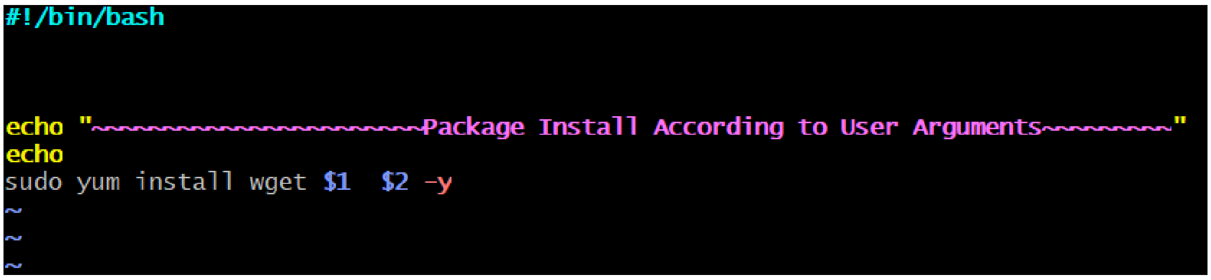
Contents and Functionality

1. **Introduction Section**
   - Displays a message indicating that the script handles package installation using arguments.

2. **Key Action**
   - Executes the yum install command with:
     - wget (predefined package in the script).
     - Additional packages passed as arguments ($1, $2).

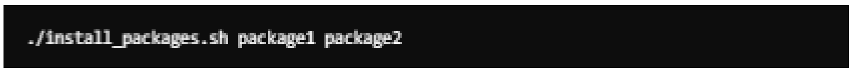
3. **Automatic Confirmation**
   - Uses the -y flag to automatically confirm installation prompts.

Input:



```
#!/bin/bash


echo "~~~~~~~~~~~~~~~~~~~~~~~~~~Package Install According to User Arguments~~~~~~~~~~"
echo
sudo yum install wget $1  $2 -y
~
~
~
```

- **Command-line arguments**: The script expects up to two additional package names passed when running the command.
  Example:



```
./install_packages.sh package1 package2
```

Here:

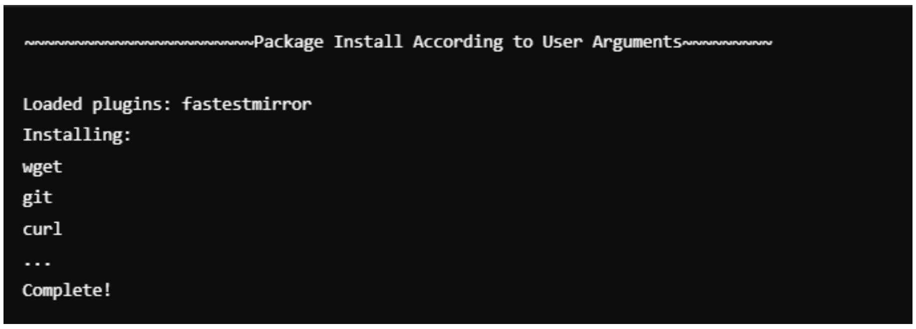
1. package1 corresponds to $1.
2. package2 corresponds to $2.

Output:

The script installs:

1. wget (default).

Packages specified as arguments ($1, $2).



```
~~~~~~~~~~~~~~~~~~~~~~~~Package Install According to User Arguments~~~~~~~~~~

Loaded plugins: fastestmirror
Installing:
wget
git
curl
...
Complete!
```

Use Case

This script is useful for:

- Simplifying the installation of multiple packages with a single command.
- Automating installations for developers or system administrators.

Automated Website Deployment Script Using Command-Line Arguments

Definition of Arguments

Arguments are values passed to a script at runtime that can be used within the script. In this script, the arguments are represented by $1, $2, and $3, which allow dynamic input for specific actions. Here, they correspond to the URL to download ($1), the file to unzip ($2), and the directory path to copy ($3).

Why We Use Arguments

Arguments make scripts flexible and reusable by allowing the user to specify different values each time the script runs. This avoids hardcoding and makes the script adaptable for various use cases without modification.

Use of the Script

This script automates the process of downloading, unzipping, and copying web files to the server's root directory by accepting URL, file, and directory as command-line arguments. This is especially useful for deploying different web applications or versions of a site on the same server setup.

Benefits

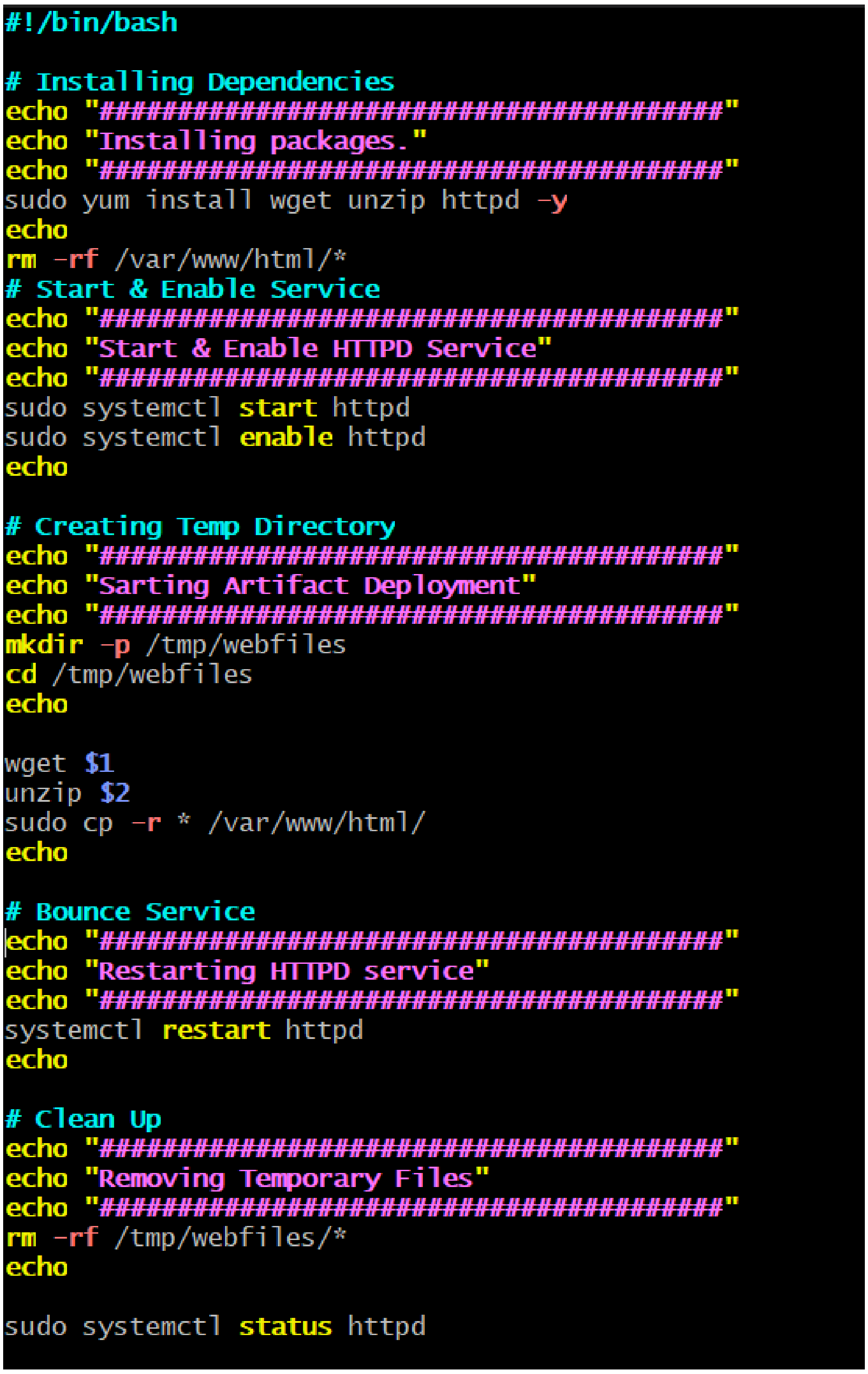
1. **Flexibility**: The script can handle various inputs, allowing it to be used with different URLs, files, and directory paths.

2. **Reusability**: The use of arguments makes it easy to reuse the script for different deployments without editing the code.

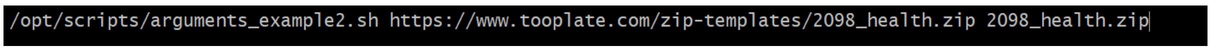
3. **Efficiency**: Speeds up deployment by automating dependency installation, file handling, and server setup.

4. **Reduced Maintenance**: Minimal changes are needed for new deployments, simplifying script maintenance and reducing the risk of errors.

Input:

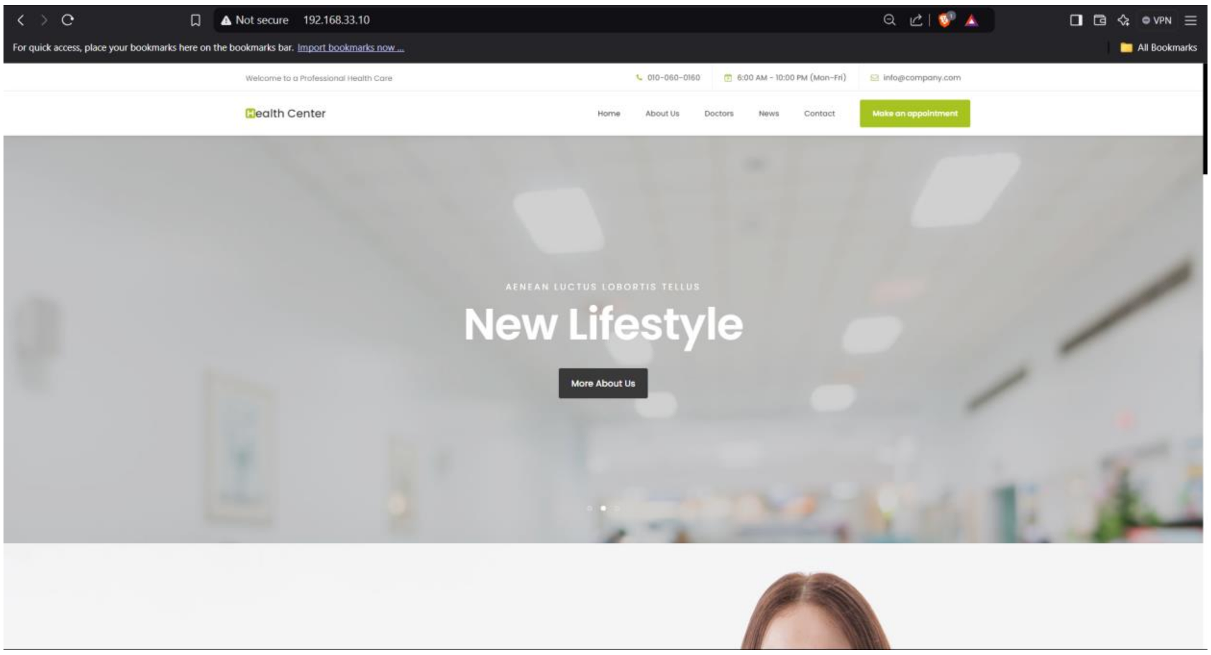


```bash
#!/bin/bash

# Installing Dependencies
echo "########################################"
echo "Installing packages."
echo "########################################"
sudo yum install wget unzip httpd -y
echo
rm -rf /var/www/html/*
# Start & Enable Service
echo "########################################"
echo "Start & Enable HTTPD Service"
echo "########################################"
sudo systemctl start httpd
sudo systemctl enable httpd
echo

# Creating Temp Directory
echo "########################################"
echo "Sarting Artifact Deployment"
echo "########################################"
mkdir -p /tmp/webfiles
cd /tmp/webfiles
echo

wget $1
unzip $2
sudo cp -r * /var/www/html/
echo

# Bounce Service
echo "########################################"
echo "Restarting HTTPD service"
echo "########################################"
systemctl restart httpd
echo

# Clean Up
echo "########################################"
echo "Removing Temporary Files"
echo "########################################"
rm -rf /tmp/webfiles/*
echo

sudo systemctl status httpd
```



```
/opt/scripts/arguments_example2.sh https://www.tooplate.com/zip-templates/2098_health.zip 2098_health.zip
```

Output:


Not secure 192.168.33.10
For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now ...
All Bookmarks
Welcome to a Professional Health Care
010-060-0160
6:00 AM - 10:00 PM (Mon-Fri)
info@company.com
Health Center
Home
About Us
Doctors
News
Contact
Make an appointment
AENEAN LUCTUS LOBORTIS TELLUS
New Lifestyle
More About Us

14-Nov-2024

Internship Day - 80 Report:

Understanding Variables in Shell Scripting

Definition

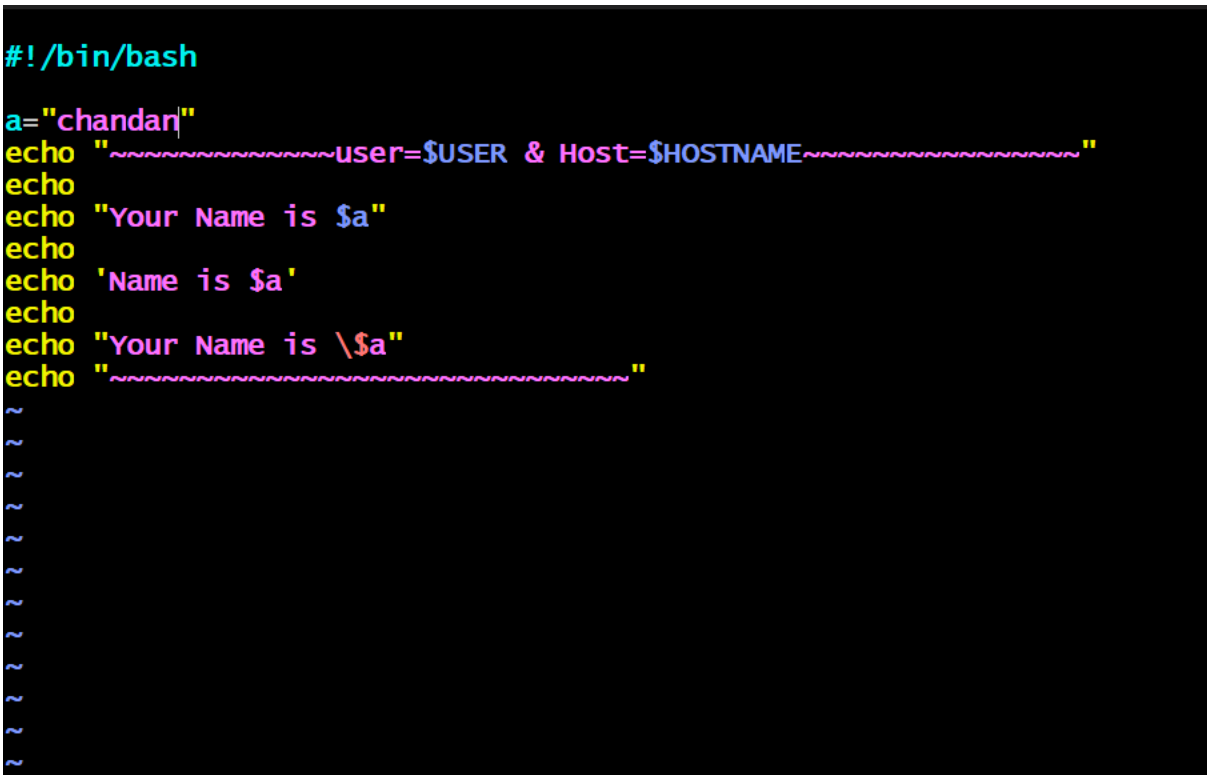
In shell scripting, variables are used to store data and reuse it throughout the script. This script demonstrates:

1. **Standard Variable Substitution**: Prints the value of a variable ($a).
2. **Literal Output of Variable Name**: Shows how to prevent variable substitution using single quotes or escape characters.

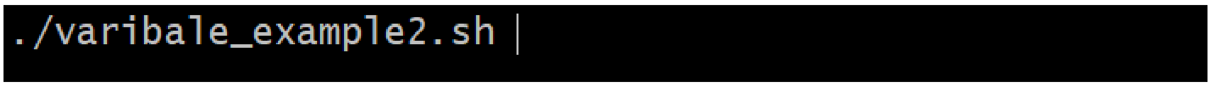
Example Breakdown:

- a="chandan" assigns the string "chandan" to the variable a.
- "Your Name is $a" displays the value of a.
- 'Name is $a' treats $a as plain text.
- "Your Name is \$a" uses the escape character (\) to print $a literally.

INPUT:

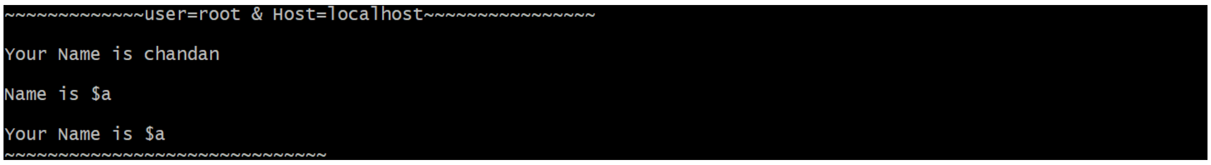


```
#!/bin/bash

a="chandan"
echo "~~~~~~~~~~~~~user=$USER & Host=$HOSTNAME~~~~~~~~~~~~~~~~"
echo
echo "Your Name is $a"
echo
echo 'Name is $a'
echo
echo "Your Name is \$a"
echo "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
~
~
~
~
~
~
~
~
~
~
~
~
```



```
./varibale_example2.sh
```

OUTPUT:



```
~~~~~~~~~~~~~user=root & Host=localhost~~~~~~~~~~~~~~~

Your Name is chandan

Name is $a

Your Name is $a
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Using if Conditions in Shell Scripting

Introduction

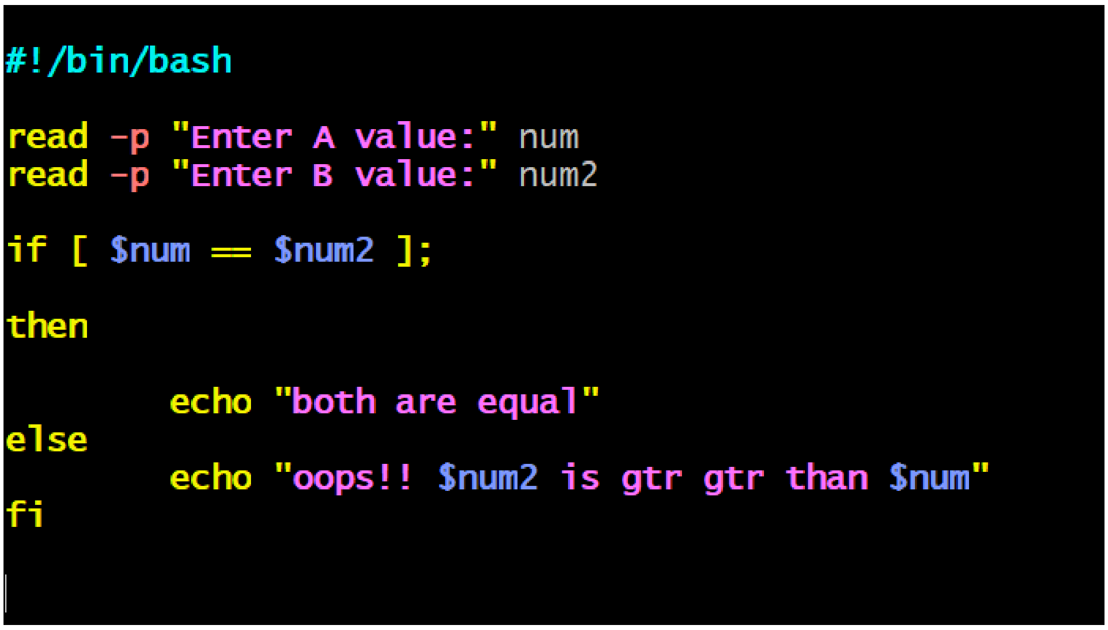
This script demonstrates the use of the if condition in shell scripting to compare two user-provided values. By reading inputs and applying a conditional check, it determines whether the values are equal or one is greater than the other, showcasing decision-making capabilities in shell scripts.

Use

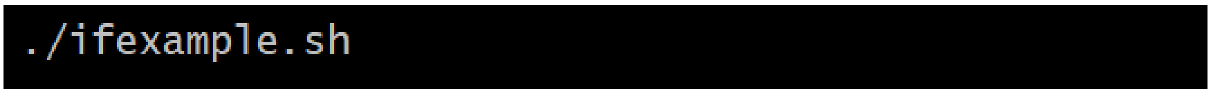
This script is useful for comparing two numerical inputs and performing conditional operations based on their values. It can be adapted for scenarios such as:

1. **Equality Checks**: Determine if two user-provided values are the same.
2. **Decision Making**: Execute specific commands or display messages based on the result of a comparison.
3. **Input Validation**: Useful in scripts where user inputs need to be compared or evaluated for further processing.

Input:

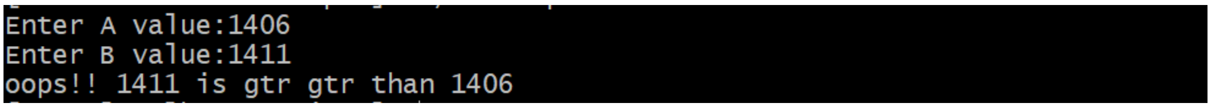


```
#!/bin/bash

read -p "Enter A value:" num
read -p "Enter B value:" num2

if [ $num == $num2 ];

then

        echo "both are equal"
else
        echo "oops!! $num2 is gtr gtr than $num"
fi
```



```
./ifexample.sh
```

Output:



```
Enter A value:1406
Enter B value:1411
oops!! 1411 is gtr gtr than 1406
```

Runtime Inputs in Shell Scripting

The uploaded file contains a shell script demonstrating how to handle runtime inputs in bash scripting. It shows the use of various read commands to capture user inputs, such as names, age, and gender, both with and without masking sensitive input.

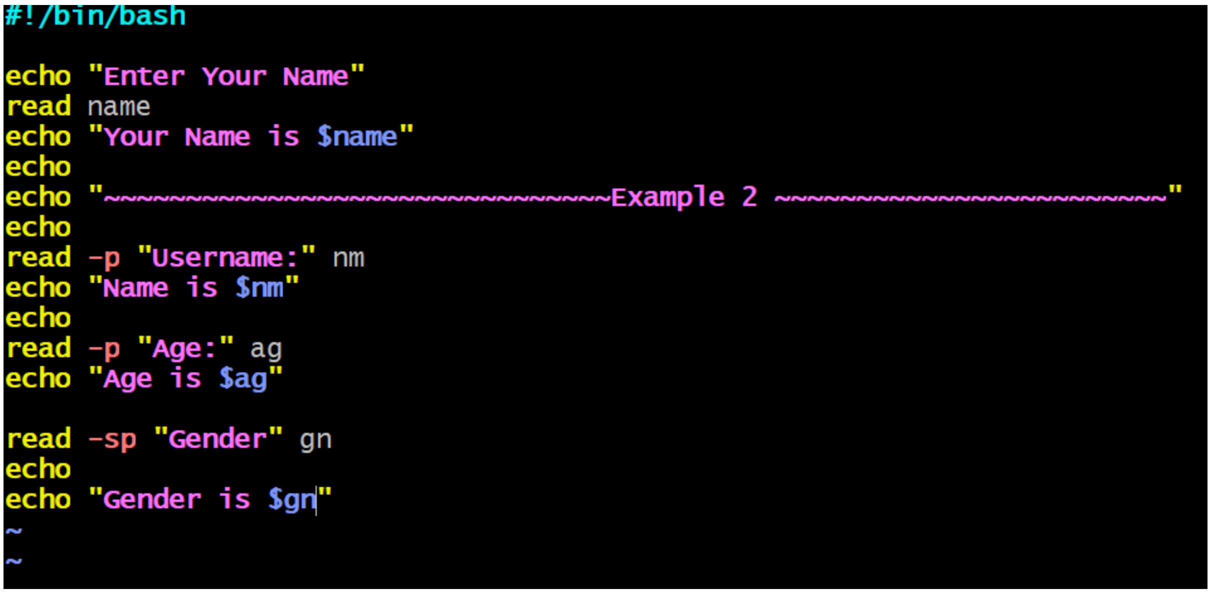
Uses of Runtime Inputs in Shell Scripting

1. **Interactive Applications**: Enables scripts to interact with users during execution.
2. **Dynamic Behavior**: Allows a script to adapt its behavior based on user-provided data.
3. **Data Collection**: Useful for capturing user-specific data like credentials, preferences, and configurations.
4. **Automation**: Facilitates parameter input for automating tasks without hardcoding values.

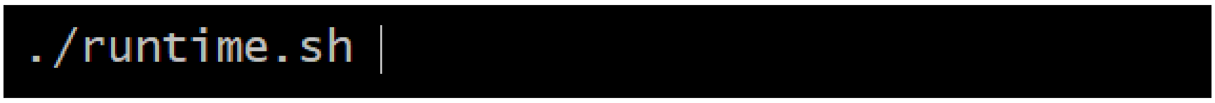
Benefits of Using Runtime Inputs

1. **Flexibility**: Eliminates the need to modify scripts for different inputs, making them reusable.
2. **User-Centric**: Increases script usability by allowing real-time user interaction.
3. **Enhanced Security**: Using masked inputs (e.g., read -sp) ensures sensitive data, like passwords, is not displayed on the screen.
4. **Ease of Debugging**: Simplifies testing by allowing varied inputs during runtime

Input:

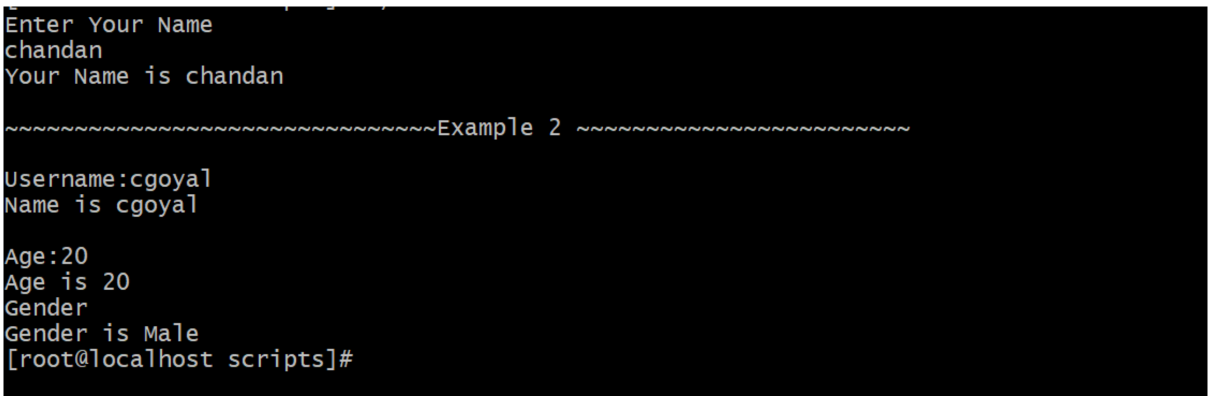


```
#!/bin/bash

echo "Enter Your Name"
read name
echo "Your Name is $name"
echo
echo "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Example 2 ~~~~~~~~~~~~~~~~~~~~~~~~~"
echo
read -p "Username:" nm
echo "Name is $nm"
echo
read -p "Age:" ag
echo "Age is $ag"

read -sp "Gender" gn
echo
echo "Gender is $gn"
~
~
```



```
./runtime.sh
```

Output:



```
Enter Your Name
chandan
Your Name is chandan

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Example 2 ~~~~~~~~~~~~~~~~~~~~~~~

Username:cgoyal
Name is cgoyal

Age:20
Age is 20
Gender
Gender is Male
[root@localhost scripts]#
```