Sure! As a Unix developer, I'll provide you with a comprehensive list of topics to learn shell scripting, along with examples to help you understand each concept better. Shell scripting is a powerful skill that allows you to automate tasks and interact with the Unix/Linux command-line environment efficiently. Let's get started:

1. Basic Shell Script Structure:

- The shebang line (#!/bin/bash) and comments

- Variables and their usage

- Command substitution

- Basic commands: echo, read, and exit

Example:

```bash

#!/bin/bash

# This is a simple shell script

# Variables

name="John"

age=25

# Command substitution

current\_date=$(date +%Y-%m-%d)

# Output

echo "Hello, $name! You are $age years old."

echo "Today's date is $current\_date"

```

2. Input and Output:

- Reading user input with read

- Redirecting output with > and >>

- Redirecting input with <

Example:

```bash

#!/bin/bash

# Read input

echo "What is your name?"

read name

# Redirecting output to a file

echo "Hello, $name!" > greeting.txt

# Appending output to a file

echo "Nice to meet you, $name." >> greeting.txt

# Redirecting input from a file

while read line; do

echo "Line: $line"

done < greeting.txt

```

3. Conditional Statements:

- if, else if, else

- test and [ ] constructs for conditions

- Logical operators: -a, -o, ! (and, or, not)

Example:

```bash

#!/bin/bash

# Get user's age

echo "Enter your age:"

read age

# Check age and provide message

if [ "$age" -lt 18 ]; then

echo "You are a minor."

elif [ "$age" -ge 18 ] && [ "$age" -lt 65 ]; then

echo "You are an adult."

else

echo "You are a senior citizen."

fi

```

4. Loops:

- for loop

- while loop

- break and continue statements

Example:

```bash

#!/bin/bash

# For loop

for i in {1..5}; do

echo "Iteration $i"

done

# While loop

count=1

while [ $count -le 5 ]; do

echo "Count: $count"

count=$((count + 1))

done

# Break and continue

for i in {1..10}; do

if [ $i -eq 3 ]; then

continue

elif [ $i -eq 8 ]; then

break

fi

echo "Value: $i"

done

```

5. Functions:

- Defining functions

- Function arguments

- Returning values from functions

Example:

```bash

#!/bin/bash

# Function definition

greet() {

local name="$1"

echo "Hello, $name!"

}

# Function call

greet "Alice"

# Function with return value

add() {

local result=$(( $1 + $2 ))

echo "$result"

}

# Function call with arguments and storing the result

sum=$(add 5 10)

echo "Sum: $sum"

```

6. Arrays:

- Creating and accessing arrays

- Looping through arrays

- Array length and slicing

Example:

```bash

#!/bin/bash

# Creating and accessing arrays

fruits=("Apple" "Banana" "Orange")

echo "First fruit: ${fruits[0]}"

echo "All fruits: ${fruits[@]}"

# Looping through arrays

for fruit in "${fruits[@]}"; do

echo "Fruit: $fruit"

done

# Array length

echo "Number of fruits: ${#fruits[@]}"

# Slicing arrays

echo "First two fruits: ${fruits[@]:0:2}"

```

7. Command-Line Arguments:

- Accessing command-line arguments

- Handling options with getopt

Example:

```bash

#!/bin/bash

# Accessing command-line arguments

echo "Script name: $0"

echo "First argument: $1"

echo "All arguments: $@"

# Using getopt for options

while getopts "abc:" opt; do

case "$opt" in

a) echo "Option a is selected." ;;

b) echo "Option b is selected." ;;

c) echo "Option c is selected with value: $OPTARG" ;;

?) echo "Invalid option: -$OPTARG" ;;

esac

done

```

8. Error Handling:

- Handling errors with exit codes

- Using "set -e" and "set +e"

Example:

```bash

#!/bin/bash

# Handling errors with exit codes

divide() {

if [ "$2" -eq 0 ]; then

echo "Error: Division by zero!"

exit 1

fi

echo "$1 / $2 = $(( $1 / $2 ))"

}

divide 10 2

divide 8 0 # This will produce an error and exit with code 1

```

9. File Operations:

- Checking file existence

- Reading and writing to files

- File permissions and ownership

Example:

```bash

#!/bin/bash

# Checking file existence

file\_path="/path/to/file.txt"

if [ -e "$file\_path" ]; then

echo "File exists!"

else

echo "File does not exist."

fi

# Reading and writing to files

echo "Hello, this is a sample file." > "$file\_path"

cat "$file\_path"

# File permissions and ownership

chmod 644 "$file\_path" # Set read/write permissions for owner, and read-only for group and others

chown user:group "$file\_path" # Change the file owner and group

```

10. Advanced Shell Techniques:

- Piping and chaining commands

- Subshells and background processes

- Using "grep," "sed," and "awk" for text processing

Example:

```bash

#!/bin/bash

# Piping and chaining commands

ls -l | grep "txt" | awk '{print $9}'

# Subshells and background processes

( sleep 5; echo "Subshell command completed" ) &

# Using grep, sed, and awk

grep "pattern" input.txt

sed 's/old/new/g' input.txt

awk '{print $2}' input.txt

```

Remember that practice is key to mastering shell scripting. Play around with these examples, try modifying them, and create your scripts to automate tasks. Shell scripting is a skill that improves with hands-on experience and exposure to real-world scenarios. Happy scripting!

Sure! Let's provide five examples with solutions for each of the sub-topics discussed earlier. We'll cover basic shell script structure, input and output, conditional statements, loops, and functions.

1. Basic Shell Script Structure:

Example 1 - Hello World:

```bash

#!/bin/bash

# This is a basic shell script printing "Hello, World!"

echo "Hello, World!"

```

Example 2 - Personalized Greeting:

```bash

#!/bin/bash

# Read user's name and provide a personalized greeting

echo "What is your name?"

read name

echo "Hello, $name! Nice to meet you."

```

2. Input and Output:

Example 3 - Saving User Input to a File:

```bash

#!/bin/bash

# Read user input and save it to a file

echo "Please enter your feedback:"

read feedback

echo "$feedback" > feedback.txt

echo "Thank you! Your feedback has been saved."

```

Example 4 - Appending to a File:

```bash

#!/bin/bash

# Ask user for a note and append it to a notes file

echo "Add a note:"

read note

echo "$note" >> notes.txt

echo "Note added successfully!"

```

3. Conditional Statements:

Example 5 - Check If a Number is Even or Odd:

```bash

#!/bin/bash

# Check if a number is even or odd

echo "Enter a number:"

read num

if [ $((num % 2)) -eq 0 ]; then

echo "$num is even."

else

echo "$num is odd."

fi

```

Example 6 - Check If a File Exists:

```bash

#!/bin/bash

# Check if a file exists

echo "Enter a file path:"

read file\_path

if [ -e "$file\_path" ]; then

echo "The file exists."

else

echo "The file does not exist."

fi

```

4. Loops:

Example 7 - Print Numbers from 1 to N:

```bash

#!/bin/bash

# Print numbers from 1 to N using a for loop

echo "Enter a number:"

read n

for ((i=1; i<=n; i++)); do

echo $i

done

```

Example 8 - Sum of Numbers using a While Loop:

```bash

#!/bin/bash

# Calculate the sum of numbers from 1 to N using a while loop

echo "Enter a number:"

read n

sum=0

count=1

while [ $count -le $n ]; do

sum=$((sum + count))

count=$((count + 1))

done

echo "Sum of numbers from 1 to $n is: $sum"

```

5. Functions:

Example 9 - Add Two Numbers using a Function:

```bash

#!/bin/bash

# Function to add two numbers

add() {

local num1=$1

local num2=$2

local result=$((num1 + num2))

echo $result

}

echo "Enter the first number:"

read a

echo "Enter the second number:"

read b

result=$(add $a $b)

echo "The sum of $a and $b is: $result"

```

Example 10 - Calculate Factorial using Recursion:

```bash

#!/bin/bash

# Function to calculate factorial using recursion

factorial() {

local num=$1

if [ $num -eq 0 ]; then

echo 1

else

local prev\_factorial=$(factorial $((num - 1)))

echo $((num \* prev\_factorial))

fi

}

echo "Enter a number to calculate its factorial:"

read num

result=$(factorial $num)

echo "Factorial of $num is: $result"

```

These examples cover a range of shell scripting topics from simple tasks like printing "Hello, World!" to more complex ones like calculating factorial using recursion. They should provide you with a good foundation for learning and practicing shell scripting at different levels of difficulty. Experiment with these examples, modify them, and try to create your own scripts to solve various problems and automate tasks. Happy scripting!

Sure! Below are brief explanations of each topic along with examples:

\*\*1. What is a shell script?\*\*

A shell script is a text file containing a series of commands that are executed in sequence by the shell interpreter.

\*\*Example:\*\*

```bash

#!/bin/bash

echo "Hello, World!"

```

\*\*2. Understanding shells (bash, sh, zsh, etc.):\*\*

Different shells have slightly different features and syntax, but they all serve the same purpose of running shell scripts.

\*\*Example:\*\*

```bash

#!/bin/zsh

echo "This is a zsh shell script."

```

\*\*3. Writing your first shell script:\*\*

Creating a simple script to print a message.

\*\*Example:\*\*

```bash

#!/bin/bash

echo "My first shell script!"

```

\*\*4. Executing shell scripts:\*\*

Running a shell script using the shell interpreter.

\*\*Example:\*\*

```bash

chmod +x my\_script.sh # Make the script executable

./my\_script.sh # Execute the script

```

\*\*5. Script file permissions and shebang (#!):\*\*

Setting executable permissions on the script file and specifying the shell to use.

\*\*Example:\*\*

```bash

#!/bin/bash

echo "This is a bash script."

```

\*\*6. Variables and data types:\*\*

Defining and using variables in shell scripts.

\*\*Example:\*\*

```bash

#!/bin/bash

name="John"

age=30

echo "My name is $name, and I am $age years old."

```

\*\*7. Command substitution and arithmetic operations:\*\*

Using command output as a variable and performing arithmetic operations.

\*\*Example:\*\*

```bash

#!/bin/bash

count=$(ls | wc -l)

echo "There are $count files in the current directory."

x=10

y=5

result=$((x + y))

echo "The sum of $x and $y is $result."

```

\*\*8. Basic input/output (read, echo):\*\*

Reading user input and printing output.

\*\*Example:\*\*

```bash

#!/bin/bash

echo "What is your name?"

read name

echo "Hello, $name!"

```

\*\*9. Conditional statements (if, else, elif):\*\*

Executing different code blocks based on conditions.

\*\*Example:\*\*

```bash

#!/bin/bash

read age

if [ $age -lt 18 ]; then

echo "You are a minor."

else

echo "You are an adult."

fi

```

\*\*10. Looping structures (for, while, until):\*\*

Repeating actions until a condition is met.

\*\*Example:\*\*

```bash

#!/bin/bash

for i in {1..5}; do

echo "Iteration $i"

done

count=0

while [ $count -lt 5 ]; do

echo "Count: $count"

((count++))

done

num=1

until [ $num -gt 5 ]; do

echo "Number: $num"

((num++))

done

```

The explanations and examples above cover the basic concepts in shell scripting. If you're interested in more intermediate and advanced concepts, please let me know!

Certainly! Let's continue with intermediate and advanced concepts in shell scripting:

\*\*11. Functions in shell scripts:\*\*

Creating and using functions to modularize code.

\*\*Example:\*\*

```bash

#!/bin/bash

function greet {

echo "Hello, $1!"

}

greet "John"

```

\*\*12. Positional parameters and command-line arguments:\*\*

Accessing arguments passed to the script or function.

\*\*Example:\*\*

```bash

#!/bin/bash

echo "Hello, $1!"

```

\*\*Usage:\*\*

```bash

./script.sh Alice

# Output: Hello, Alice!

```

\*\*13. Arrays and associative arrays:\*\*

Working with arrays and key-value pairs.

\*\*Example:\*\*

```bash

#!/bin/bash

fruits=("Apple" "Banana" "Orange")

echo "I like ${fruits[0]} and ${fruits[1]}."

declare -A colors

colors["Apple"]="Red"

colors["Banana"]="Yellow"

echo "The color of Apple is ${colors["Apple"]}."

```

\*\*14. String manipulation and regular expressions:\*\*

Manipulating strings and using regular expressions.

\*\*Example:\*\*

```bash

#!/bin/bash

str="Hello, World!"

echo ${str:0:5} # Output: Hello

echo ${str,,} # Output: hello, world!

```

\*\*15. File handling (reading, writing, appending):\*\*

Reading from, writing to, and appending to files.

\*\*Example:\*\*

```bash

#!/bin/bash

echo "Line 1" > file.txt # Write to file

echo "Line 2" >> file.txt # Append to file

cat file.txt # Read from file

```

\*\*16. Command-line options and getopt:\*\*

Handling command-line options in shell scripts.

\*\*Example:\*\*

```bash

#!/bin/bash

while getopts ":abc" opt; do

case $opt in

a) echo "Option a is selected." ;;

b) echo "Option b is selected." ;;

c) echo "Option c is selected." ;;

\?) echo "Invalid option: -$OPTARG" ;;

esac

done

```

\*\*Usage:\*\*

```bash

./script.sh -ab

# Output: Option a is selected. Option b is selected.

```

\*\*17. Working with files and directories (File tests, find, grep, etc.):\*\*

Performing file tests, finding files, and searching in files.

\*\*Example:\*\*

```bash

#!/bin/bash

if [ -f "file.txt" ]; then

echo "file.txt exists and is a regular file."

fi

find . -type f -name "\*.txt"

grep "pattern" file.txt

```

\*\*18. Error handling (exit codes, traps, error handling techniques):\*\*

Dealing with errors in shell scripts.

\*\*Example:\*\*

```bash

#!/bin/bash

set -e # Exit on error

command\_that\_might\_fail

echo "This line won't be reached if the previous command fails."

trap "echo Error occurred!" ERR

command\_that\_might\_fail

```

\*\*19. Process handling (background, foreground, job control):\*\*

Managing background and foreground processes.

\*\*Example:\*\*

```bash

#!/bin/bash

sleep 10 & # Run a command in the background

echo "Process ID: $!"

wait # Wait for all background jobs to finish

```

\*\*20. Signals and signal handling:\*\*

Handling signals sent to the script.

\*\*Example:\*\*

```bash

#!/bin/bash

function sigint\_handler {

echo "Received SIGINT. Exiting."

exit 1

}

trap sigint\_handler SIGINT

while true; do

echo "Script is running..."

sleep 1

done

```

These intermediate and advanced topics will significantly expand your shell scripting skills. Continue exploring and practicing them to become a proficient shell script developer.

Sure! Let's dive into the details and examples for each topic:

\*\*19. Process handling (background, foreground, job control):\*\*

- Background processes: You can run a command in the background by appending an ampersand (&) at the end of the command. The script will continue running without waiting for the background process to complete.

\*\*Example:\*\*

```bash

#!/bin/bash

sleep 10 & # Run a command in the background

echo "Script continues while sleep is running."

```

- Foreground processes: When a command is executed without an ampersand, it runs in the foreground, and the script waits for it to finish before proceeding to the next command.

\*\*Example:\*\*

```bash

#!/bin/bash

echo "Starting a long process..."

sleep 10 # Run a command in the foreground

echo "Process completed."

```

- Job control: You can manage background processes using job control commands like `fg` (bring a process to the foreground) and `bg` (send a stopped process to the background). The `jobs` command lists all background jobs.

\*\*Example:\*\*

```bash

#!/bin/bash

sleep 10 & # Run a command in the background

bg # Send the background job to the background

jobs # List all background jobs

```

\*\*Signals and signal handling:\*\*

Signals are software interrupts sent to a process to notify it of an event. Shell scripts can handle signals using the `trap` command to execute a custom action when a specific signal is received.

\*\*Example:\*\*

```bash

#!/bin/bash

function sigint\_handler {

echo "Received SIGINT. Exiting."

exit 1

}

trap sigint\_handler SIGINT

while true; do

echo "Script is running..."

sleep 1

done

```

In this example, the script sets up a signal handler for SIGINT (Ctrl+C), and when the script receives this signal, it will execute the `sigint\_handler` function before exiting.

\*\*Process substitution and pipelines:\*\*

Process substitution allows you to use the output of a command as a file input or output for another command.

\*\*Example:\*\*

```bash

#!/bin/bash

# Using process substitution to pass the output of a command as input to another command

diff <(sort file1.txt) <(sort file2.txt)

```

Pipelines are used to connect multiple commands, where the output of one command becomes the input of the next command.

\*\*Example:\*\*

```bash

#!/bin/bash

# Using a pipeline to count the number of lines in a file

cat file.txt | wc -l

```

\*\*Advanced command-line parsing (getopts):\*\*

The `getopts` command allows you to parse command-line options and arguments in a shell script.

\*\*Example:\*\*

```bash

#!/bin/bash

while getopts ":abc" opt; do

case $opt in

a) echo "Option a is selected." ;;

b) echo "Option b is selected." ;;

c) echo "Option c is selected." ;;

\?) echo "Invalid option: -$OPTARG" ;;

esac

done

```

\*\*Usage:\*\*

```bash

./script.sh -ab

# Output: Option a is selected. Option b is selected.

```

\*\*Advanced string manipulation and pattern matching (sed, awk):\*\*

`sed` and `awk` are powerful text-processing tools used for pattern matching, search and replace, and more.

\*\*Example:\*\*

```bash

#!/bin/bash

# Using sed to replace a word in a file

sed -i 's/old\_word/new\_word/g' file.txt

```

\*\*Text processing tools (grep, cut, sort, uniq):\*\*

`grep` is used for searching text patterns, `cut` is used for extracting specific columns, `sort` is used for sorting lines, and `uniq` is used for removing duplicate lines.

\*\*Example:\*\*

```bash

#!/bin/bash

# Using grep to search for lines containing a specific pattern

grep "error" log.txt

# Using cut to extract the second column from a CSV file

cut -d ',' -f 2 data.csv

# Using sort and uniq to sort and remove duplicate lines

sort file.txt | uniq

```

\*\*Working with streams and redirects:\*\*

In Unix, everything is treated as a file. Standard Input (stdin), Standard Output (stdout), and Standard Error (stderr) are streams that can be redirected.

\*\*Example:\*\*

```bash

#!/bin/bash

# Redirecting stdout to a file

echo "Hello, World!" > output.txt

# Appending stdout to a file

echo "This is a new line." >> output.txt

# Redirecting stderr to a file

command\_not\_found 2> error.log

```

\*\*Advanced file handling (file permissions, file descriptors):\*\*

Understanding and manipulating file permissions and file descriptors (file handles).

\*\*Example:\*\*

```bash

#!/bin/bash

# Changing file permissions

chmod +x script.sh

# Redirecting input from a file descriptor

command < input.txt

# Redirecting output to a file descriptor

command > output.txt

```

\*\*Subshells and process substitution:\*\*

Subshells are used to run commands in a separate shell environment.

\*\*Example:\*\*

```bash

#!/bin/bash

# Using a subshell to set a variable that doesn't affect the parent shell

(sub\_var=$(ls); echo "Inside subshell: $sub\_var")

# The variable is not accessible here

echo "Outside subshell: $sub\_var"

```

\*\*Debugging shell scripts (set -x, set -e):\*\*

Using `set -x` to enable debugging mode, which prints each command before execution, and `set -e` to exit immediately if any command returns a non-zero exit code.

\*\*Example:\*\*

```bash

#!/bin/bash

set -x # Enable debugging mode

set -e # Exit on error

command\_that\_might\_fail

echo "This line won't be reached if the previous command fails."

```

\*\*Best Practices and Optimization:\*\*

\*\*29. Writing clean and maintainable code:\*\*

Writing clear, well-organized, and easy-to-understand code with proper indentation and meaningful variable names.

\*\*Shellcheck - A tool for shell script linting:\*\*

Shellcheck is a static analysis tool that helps identify syntax errors and potential issues in shell scripts.

\*\*Shell scripting style guide:\*\*

Following a style guide to maintain consistency in coding practices, like Google's Shell Style Guide.

\*\*Using functions effectively for code reusability:\*\*

Modularizing code into functions for easier maintenance and reuse.

\*\*Performance optimization tips and techniques:\*\*

Optimizing shell scripts for efficiency, like avoiding unnecessary loops and minimizing process overhead.

\*\*Real-World Examples and Projects:\*\*

\*\*34. Writing simple automation scripts (e.g., backup scripts):\*\*

Automating routine tasks like backups using shell scripts.

\*\*System administration scripts (e.g., user management):\*\*

Creating scripts to manage users, groups, and permissions on a system.

\*\*Log parsing and analysis scripts:\*\*

Parsing log files to extract useful information or perform analysis.

\*\*Web scraping with shell scripts:\*\*

Extracting data from websites using command-line tools like `curl` and `grep`.

\*\*Network monitoring scripts:\*\*

Monitoring network resources, like pinging hosts or checking open ports.

\*\*Shell scripting in DevOps and CI/

CD workflows:\*\*

Using shell scripts to automate tasks in the DevOps and CI/CD pipelines.

Mastering these advanced concepts, best practices, and real-world examples will make you a proficient shell script developer. Practice and real-world application are key to becoming more skilled in shell scripting.

Best Practices and Optimization:

1. Writing clean and maintainable code:

Example:

```bash

#!/bin/bash

# This script prints "Hello, World!" to the console.

function print\_hello() {

local message="Hello, World!"

echo "$message"

}

print\_hello

```

2. Shellcheck - A tool for shell script linting:

Example:

```bash

#!/bin/bash

# This script demonstrates a syntax error identified by Shellcheck.

variable = "Hello" # Space not allowed before '='

echo "$variable"

```

3. Shell scripting style guide:

Example:

```bash

#!/bin/bash

# This script follows Google's Shell Style Guide.

function greet\_user() {

local name="$1"

echo "Hello, $name!"

}

greet\_user "John"

```

4. Using functions effectively for code reusability:

Example:

```bash

#!/bin/bash

# This script demonstrates code modularity with functions.

function calculate\_sum() {

local a=$1

local b=$2

echo $((a + b))

}

result=$(calculate\_sum 10 5)

echo "The sum is: $result"

```

5. Performance optimization tips and techniques:

Example:

```bash

#!/bin/bash

# This script avoids unnecessary loops by using arithmetic expansion.

start=1

end=1000000

# Slow approach with 'for' loop

for ((i=start; i<=end; i++)); do

echo $i

done

# Faster approach with arithmetic expansion

seq $start $end

```

Real-World Examples and Projects:

6. Writing simple automation scripts (e.g., backup scripts):

Example:

```bash

#!/bin/bash

# This script backs up important files to a specified directory.

backup\_dir="/path/to/backup"

important\_files="/path/to/files/\*"

cp -r $important\_files $backup\_dir

```

7. System administration scripts (e.g., user management):

Example:

```bash

#!/bin/bash

# This script creates a new user and assigns them to a specified group.

username="newuser"

group="users"

useradd $username

usermod -a -G $group $username

echo "User $username created and added to group $group."

```

8. Log parsing and analysis scripts:

Example:

```bash

#!/bin/bash

# This script parses a log file and counts occurrences of specific events.

log\_file="/path/to/logfile.log"

# Count occurrences of "error" in the log file

error\_count=$(grep -c "error" $log\_file)

echo "Number of errors in the log: $error\_count"

```

9. Web scraping with shell scripts:

Example:

```bash

#!/bin/bash

# This script extracts the titles of articles from a webpage.

url="https://example.com/articles"

# Use curl to fetch the webpage content and grep to extract titles

curl -s $url | grep -o '<h2>.\*</h2>' | sed 's/<[^>]\*>//g'

```

10. Network monitoring scripts:

Example:

```bash

#!/bin/bash

# This script pings a list of hosts to check their availability.

hosts=("google.com" "example.com" "localhost")

for host in "${hosts[@]}"; do

if ping -c 1 $host &> /dev/null; then

echo "$host is reachable."

else

echo "$host is unreachable."

fi

done

```

11. Shell scripting in DevOps and CI/CD workflows:

Example:

```bash

#!/bin/bash

# This script automates the build and deployment process in a CI/CD pipeline.

echo "Running tests..."

# Code to run tests here

echo "Building the application..."

# Code to build the application here

echo "Deploying the application..."

# Code to deploy the application here

```

These examples showcase various aspects of shell scripting, from best practices and optimization techniques to real-world projects. Keep in mind that shell scripting can be powerful and versatile, but always prioritize security and handle user inputs with care, especially when executing commands with elevated privileges.

As an AI language model, I can provide you with 20 shell script examples that cover various real-time DevOps scenarios. Keep in mind that these scripts might not be complete solutions for your specific use case, but they should serve as a starting point. You can modify and expand them to suit your needs.

1. \*\*Git Pull and Auto-Deploy Script:\*\*

This script pulls the latest code from a Git repository and triggers an automatic deployment.

```bash

#!/bin/bash

REPO\_PATH="/path/to/your/repo"

DEPLOY\_PATH="/path/to/your/deployment"

cd "$REPO\_PATH"

git pull origin master

cp -r \* "$DEPLOY\_PATH"

echo "Deployment completed!"

```

2. \*\*Backup Script for MySQL Database:\*\*

This script performs a backup of a MySQL database and compresses the output.

```bash

#!/bin/bash

DB\_USER="your\_db\_user"

DB\_PASS="your\_db\_password"

DB\_NAME="your\_db\_name"

BACKUP\_DIR="/path/to/backup/directory"

mysqldump -u "$DB\_USER" -p"$DB\_PASS" "$DB\_NAME" | gzip > "$BACKUP\_DIR/db\_backup\_$(date '+%Y%m%d').sql.gz"

echo "Database backup completed!"

```

3. \*\*Log Rotation Script:\*\*

This script rotates log files and keeps a specific number of backups.

```bash

#!/bin/bash

LOG\_FILE="/path/to/your/log/file.log"

MAX\_BACKUPS=5

if [ -f "$LOG\_FILE" ]; then

mv "$LOG\_FILE" "$LOG\_FILE.$(date '+%Y%m%d')"

find "$(dirname "$LOG\_FILE")" -name "$(basename "$LOG\_FILE").\*" | sort -r | tail -n +$((MAX\_BACKUPS+1)) | xargs rm -f

fi

touch "$LOG\_FILE"

echo "Log rotation completed!"

```

4. \*\*Deployment Rollback Script:\*\*

This script rolls back a deployment to a previous version.

```bash

#!/bin/bash

DEPLOY\_PATH="/path/to/your/deployment"

BACKUP\_DIR="/path/to/backup/directory"

if [ -d "$BACKUP\_DIR" ]; then

latest\_backup=$(ls -t "$BACKUP\_DIR" | head -n 1)

cp -r "$BACKUP\_DIR/$latest\_backup" "$DEPLOY\_PATH"

echo "Rollback to $latest\_backup completed!"

else

echo "No backups found for rollback."

fi

```

5. \*\*Docker Container Cleanup Script:\*\*

This script stops and removes all Docker containers and associated volumes.

```bash

#!/bin/bash

docker stop $(docker ps -aq)

docker rm $(docker ps -aq)

docker volume rm $(docker volume ls -q)

echo "Docker container cleanup completed!"

```

6. \*\*Deployment Status Check Script:\*\*

This script checks the status of a deployed service.

```bash

#!/bin/bash

SERVICE\_URL="http://your-service-url"

TIMEOUT=10

response=$(curl -s -o /dev/null -w "%{http\_code}" "$SERVICE\_URL" --max-time $TIMEOUT)

if [ "$response" -eq 200 ]; then

echo "Service is up and running."

else

echo "Service is not responding properly (HTTP status code: $response)."

fi

```

7. \*\*SSH Key Generator Script:\*\*

This script generates an SSH key pair for passwordless authentication.

```bash

#!/bin/bash

SSH\_DIR="$HOME/.ssh"

KEY\_NAME="your\_key\_name"

mkdir -p "$SSH\_DIR"

ssh-keygen -t rsa -b 4096 -C "$KEY\_NAME" -f "$SSH\_DIR/$KEY\_NAME"

echo "SSH key pair generated in $SSH\_DIR!"

```

8. \*\*Disk Space Monitoring Script:\*\*

This script checks disk usage and sends an alert if it exceeds a specified threshold.

```bash

#!/bin/bash

THRESHOLD\_PERCENT=90

used\_percent=$(df -h | awk '$NF=="/"{print $5}' | sed 's/%//')

if [ "$used\_percent" -ge "$THRESHOLD\_PERCENT" ]; then

echo "Disk space usage is above $THRESHOLD\_PERCENT%."

# Add alerting logic here (e.g., send an email or notification).

else

echo "Disk space usage is within limits."

fi

```

9. \*\*Nginx Configuration Validation Script:\*\*

This script checks the validity of Nginx configuration files.

```bash

#!/bin/bash

NGINX\_CONF="/etc/nginx/nginx.conf"

nginx -t -c "$NGINX\_CONF"

```

10. \*\*Service Restart Script:\*\*

This script restarts a service (e.g., Nginx) after updating its configuration.

```bash

#!/bin/bash

SERVICE="nginx"

systemctl restart "$SERVICE"

echo "$SERVICE restarted!"

```

11. \*\*SSL Certificate Renewal Script:\*\*

This script renews SSL certificates using Certbot (for Let's Encrypt).

```bash

#!/bin/bash

certbot renew

```

12. \*\*Build and Push Docker Image Script:\*\*

This script builds a Docker image and pushes it to a container registry.

```bash

#!/bin/bash

IMAGE\_NAME="your\_image\_name"

IMAGE\_TAG="latest"

DOCKERFILE\_PATH="/path/to/your/Dockerfile"

REGISTRY\_URL="your.registry.com"

docker build -t "$IMAGE\_NAME:$IMAGE\_TAG" -f "$DOCKERFILE\_PATH" .

docker tag "$IMAGE\_NAME:$IMAGE\_TAG" "$REGISTRY\_URL/$IMAGE\_NAME:$IMAGE\_TAG"

docker push "$REGISTRY\_URL/$IMAGE\_NAME:$IMAGE\_TAG"

echo "Docker image $IMAGE\_NAME:$IMAGE\_TAG pushed to $REGISTRY\_URL!"

```

13. \*\*AWS S3 File Upload Script:\*\*

This script uploads a file to an AWS S3 bucket using AWS CLI.

```bash

#!/bin/bash

AWS\_PROFILE="your\_aws\_profile"

S3\_BUCKET="your\_bucket\_name"

FILE\_PATH="/path/to/your/file.txt"

S3\_KEY="path/to/destination/in/s3/file.txt"

aws --profile "$AWS\_PROFILE" s3 cp "$FILE\_PATH" "s3://$S3\_BUCKET/$S3\_KEY"

echo "File uploaded to S3!"

```

14. \*\*AWS EC2 Instance Provisioning Script:\*\*

This script provisions an AWS EC2 instance using AWS CLI.

```bash

#!/bin/bash

AWS\_PROFILE="your\_aws\_profile"

INSTANCE\_TYPE="t2.micro"

AMI\_ID="your\_ami\_id"

KEY\_NAME="your\_key\_name"

SECURITY\_GROUP="your\_security\_group"

SUBNET\_ID="your\_subnet\_id"

instance\_id=$(aws --profile "$AWS\_PROFILE" ec2 run-instances \

--image-id "$AMI\_ID" \

--instance-type "$INSTANCE\_TYPE" \

--key-name "$KEY\_NAME" \

--security-group-ids "$SECURITY\_GROUP" \

--subnet-id "$SUBNET\_ID" \

--query 'Instances[0].InstanceId' \

--output text)

echo "EC2 instance $instance\_id created!"

```

15. \*\*Jenkins Job Trigger Script:\*\*

This script triggers a Jenkins job using the Jenkins CLI.

```bash

#!/bin/bash

JENKINS\_URL="http://your\_jenkins\_url"

JOB\_NAME="your\_job\_name"

JENKINS\_USERNAME="your\_jenkins\_username"

JENKINS\_TOKEN="your\_jenkins\_api\_token"

java

-jar jenkins-cli.jar -s "$JENKINS\_URL" -auth "$JENKINS\_USERNAME:$JENKINS\_TOKEN" build "$JOB\_NAME"

echo "Jenkins job $JOB\_NAME triggered!"

```

16. \*\*Kubernetes Deployment Script:\*\*

This script deploys a Kubernetes manifest using kubectl.

```bash

#!/bin/bash

MANIFEST\_FILE="your\_kubernetes\_manifest.yaml"

kubectl apply -f "$MANIFEST\_FILE"

echo "Kubernetes deployment completed!"

```

17. \*\*Log Streaming Script:\*\*

This script streams the logs of a running container.

```bash

#!/bin/bash

CONTAINER\_NAME="your\_container\_name"

docker logs -f "$CONTAINER\_NAME"

```

18. \*\*Elasticsearch Index Cleanup Script:\*\*

This script deletes old Elasticsearch indices.

```bash

#!/bin/bash

ELASTICSEARCH\_HOST="your\_elasticsearch\_host"

INDEX\_PREFIX="your\_index\_prefix"

DAYS\_TO\_KEEP=30

indices=$(curl -s "$ELASTICSEARCH\_HOST/\_cat/indices/$INDEX\_PREFIX-\*" | awk -v days="$DAYS\_TO\_KEEP" '$7 < strftime("%Y.%m.%d", systime() - days \* 86400) {print $3}')

for index in $indices; do

curl -X DELETE "$ELASTICSEARCH\_HOST/$index"

done

echo "Elasticsearch index cleanup completed!"

```

19. \*\*Service Monitoring Script:\*\*

This script checks the status of a service and restarts it if necessary.

```bash

#!/bin/bash

SERVICE\_NAME="your\_service\_name"

systemctl is-active --quiet "$SERVICE\_NAME" || systemctl start "$SERVICE\_NAME"

echo "Service $SERVICE\_NAME is running."

```

20. \*\*Remote Server Connection Script:\*\*

This script connects to a remote server using SSH.

```bash

#!/bin/bash

REMOTE\_SERVER="your\_remote\_server"

REMOTE\_USER="your\_remote\_user"

ssh "$REMOTE\_USER"@"$REMOTE\_SERVER"

```

Remember to make the scripts executable using `chmod +x script\_name.sh` before running them. Also, ensure that you have the necessary permissions and configurations in place to execute these scripts successfully in your environment.