Azure Bash Scripting Tutorial

Author: Aditya Saxena

Date: June 2025

# PRE-SCRIPTING COMMANDS

# Introduction to Basic Bash Commands

# Author: Aditya Saxena

# Format: Plaintext

# Section: Pre-Scripting Foundation

# Date: June 2025

=============================================================

SECTION 0: BASIC BASH COMMANDS FOR BEGINNERS

=============================================================

Bash is both a shell (command-line interface) and a scripting language.

The shell allows users to interact with the operating system by typing commands.

-------------------------------------------------------------

1. NAVIGATION COMMANDS

-------------------------------------------------------------

pwd # Print working directory

ls # List files and directories

ls -l # Long listing with details

cd <directory> # Change directory

cd .. # Move up one directory level

cd ~ # Go to home directory

-------------------------------------------------------------

2. FILE AND DIRECTORY MANAGEMENT

-------------------------------------------------------------

touch file.txt # Create an empty file

mkdir new\_folder # Create a new directory

cp file1 file2 # Copy file1 to file2

mv file1 file2 # Rename or move file1 to file2

rm file.txt # Delete a file

rm -r dir/ # Delete a directory and its contents

-------------------------------------------------------------

3. VIEWING FILE CONTENT

-------------------------------------------------------------

cat file.txt # View entire file

less file.txt # View file page by page (q to quit)

head file.txt # First 10 lines

tail file.txt # Last 10 lines

tail -f file.txt # Live monitoring (e.g., logs)

-------------------------------------------------------------

4. FILE PERMISSIONS

-------------------------------------------------------------

In Unix/Linux systems, every file and directory has associated \*\*permissions\*\* that control who can read, write, or execute them.

-------------------------------------------------------------

A. VIEWING PERMISSIONS

-------------------------------------------------------------

Command:

ls -l filename

Example output:

-rwxr-xr-- 1 aditya staff 1234 Jun 17 09:00 script.sh

Explanation of parts:

- First character (`-`): Type of file

- `-` regular file

- `d` directory

- `l` symbolic link

- Next 9 characters (`rwxr-xr--`) are permissions grouped as:

- `rwx` → Owner (user)

- `r-x` → Group

- `r--` → Others

Meaning:

- `r` = read

- `w` = write

- `x` = execute

- `-` = no permission

-------------------------------------------------------------

B. CHANGING PERMISSIONS WITH `chmod`

-------------------------------------------------------------

Basic usage:

chmod <permissions> <filename>

Examples:

chmod +x script.sh # Add execute permission

chmod -w notes.txt # Remove write permission

chmod u+x file.sh # Add execute permission for user only

Symbolic format:

- `u` = user (owner)

- `g` = group

- `o` = others

- `a` = all (u+g+o)

Examples:

chmod u+x,g-w,o-r file.txt

chmod a+r file.txt

-------------------------------------------------------------

C. OCTAL PERMISSION NOTATION

-------------------------------------------------------------

Each permission set is represented by a number:

- Read (r) = 4

- Write (w) = 2

- Execute (x) = 1

Combine the numbers:

- `7` = rwx (4+2+1)

- `6` = rw- (4+2)

- `5` = r-x (4+1)

- `4` = r-- (4)

- `0` = --- (no permission)

Examples:

chmod 755 script.sh # rwxr-xr-x (user: all, group: read/exec, others: read/exec)

chmod 644 file.txt # rw-r--r-- (standard for readable text files)

chmod 700 secrets.sh # rwx------ (private script)

-------------------------------------------------------------

D. CHANGING FILE OWNERSHIP WITH `chown`

-------------------------------------------------------------

chown newuser file.txt # Change owner

chown newuser:newgroup file.txt # Change owner and group

-------------------------------------------------------------

E. SPECIAL PERMISSION BITS (Advanced)

-------------------------------------------------------------

1. \*\*Setuid (s)\*\* – Executes file with owner's permissions

chmod u+s file.sh → will run with owner's privileges

2. \*\*Setgid (s)\*\* – Executes directory/file with group’s permissions

chmod g+s dir/

3. \*\*Sticky bit (t)\*\* – Files in directory can only be deleted by owner

chmod +t /shared

Example of sticky bit usage:

ls -ld /tmp

drwxrwxrwt 10 root root 4096 Jun 17 10:30 /tmp

-------------------------------------------------------------

F. BEST PRACTICE RECOMMENDATIONS

-------------------------------------------------------------

- Scripts: chmod 755 (rwxr-xr-x)

- Config files: chmod 644 (rw-r--r--)

- Private keys/secrets: chmod 600 (rw-------)

- Shared directories: use sticky bit (chmod +t)

- Never use chmod 777 unless absolutely necessary (gives full access to everyone)

-------------------------------------------------------------

END OF FILE PERMISSIONS SECTION

-------------------------------------------------------------

-------------------------------------------------------------

5. SYSTEM INFORMATION

-------------------------------------------------------------

whoami # Current username

hostname # Machine name

uptime # How long system has been running

df -h # Disk usage (human-readable)

free -h # RAM usage

-------------------------------------------------------------

6. PROCESS MANAGEMENT

-------------------------------------------------------------

ps # View running processes

top # Real-time process monitor

kill <pid> # Terminate a process

jobs # Background jobs

fg # Bring job to foreground

bg # Resume job in background

-------------------------------------------------------------

7. NETWORKING

-------------------------------------------------------------

ping google.com # Test network connection

ifconfig / ip a # Show IP address

curl url # Fetch webpage data

wget url # Download files

-------------------------------------------------------------

8. TEXT UTILITIES

-------------------------------------------------------------

echo "text" # Print text to screen

date # Current date/time

wc file.txt # Word/line/character count

sort file.txt # Sort file lines

uniq file.txt # Remove duplicate lines

grep "word" file # Search for word in file

-------------------------------------------------------------

9. PACKAGE MANAGEMENT (VARIES BY OS)

-------------------------------------------------------------

# Ubuntu/Debian:

sudo apt update

sudo apt install <package>

# RHEL/CentOS:

sudo yum install <package>

-------------------------------------------------------------

10. GETTING HELP

-------------------------------------------------------------

man <command> # Manual page (e.g., man ls)

<command> --help # Basic usage summary

=============================================================

END OF BASIC COMMANDS

# SCRIPTING

# Bash Scripting Theory Guide

# Author: Aditya Saxena

# Format: Plaintext

# Date: June 2025

====================================================================

SECTION 1: INTRODUCTION TO BASH

====================================================================

- Bash stands for "Bourne Again Shell"

- It is a Unix shell and command language used in Linux systems

- Commonly used for automating tasks, managing servers, and building DevOps pipelines

Usage:

$ bash script.sh # Execute a script

$ ./script.sh # Execute if script is marked executable

====================================================================

SECTION 2: BASIC SCRIPT STRUCTURE

====================================================================

Every Bash script starts with a shebang line:

#!/bin/bash

Example:

#!/bin/bash

echo "Hello, World!"

date

Save this as `hello.sh` and run using:

$ chmod +x hello.sh

$ ./hello.sh

====================================================================

SECTION 3: VARIABLES AND DATA TYPES

====================================================================

Syntax:

VARIABLE\_NAME=value

Example:

name="Aditya"

echo "Hello, $name"

Note:

- No spaces around `=`

- Use `$` to reference variables

====================================================================

SECTION 4: INPUT AND OUTPUT

====================================================================

Read input:

read -p "Enter your name: " name

echo "Hello, $name"

Redirection:

> : redirect stdout (overwrite)

>> : redirect stdout (append)

2> : redirect stderr

| : pipe output of one command into another

====================================================================

SECTION 5: CONDITIONAL STATEMENTS

====================================================================

if [ condition ]; then

# code

elif [ condition ]; then

# code

else

# code

fi

Example:

if [ $age -ge 18 ]; then

echo "Adult"

else

echo "Minor"

fi

Operators:

-eq, -ne, -lt, -le, -gt, -ge, ==, !=, -z (empty string)

====================================================================

SECTION 6: LOOPS

====================================================================

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++))

done

Until loop:

until [ $count -gt 5 ]; do

echo "Count: $count"

((count++))

done

====================================================================

SECTION 7: FUNCTIONS

====================================================================

function greet() {

echo "Hello, $1"

}

greet "Aditya"

Return values:

function add() {

return $(($1 + $2))

}

add 2 3

echo $? # Return value

====================================================================

SECTION 8: ARRAYS

====================================================================

Declare array:

arr=("apple" "banana" "cherry")

echo ${arr[1]} # Access second element

echo ${#arr[@]} # Length

for item in "${arr[@]}"; do

echo $item

done

====================================================================

SECTION 9: STRING MANIPULATION

====================================================================

str="hello world"

echo ${str:0:5} # Substring

echo ${#str} # Length

echo ${str/world/bash} # Replace

====================================================================

SECTION 10: FILE TESTS

====================================================================

if [ -f filename ]; then # File exists

if [ -d dirname ]; then # Directory exists

if [ -r file ]; then # Readable

if [ -w file ]; then # Writable

if [ -x file ]; then # Executable

====================================================================

SECTION 11: SCRIPT DEBUGGING

====================================================================

- Use `bash -x script.sh` to trace execution

- Use `set -x` inside script for line-by-line trace

- Use `set -e` to exit on error

====================================================================

SECTION 12: ADVANCED TOPICS

====================================================================

1. HEREDOC:

cat <<EOF

Multi-line text

goes here

EOF

2. TRAP for cleanup:

trap "echo 'Script interrupted'; exit" SIGINT

3. Command substitution:

current\_date=$(date)

4. Subshells:

(output=$(ls -l)) # Isolated process

5. Scheduling with cron:

$ crontab -e

0 \* \* \* \* /path/to/script.sh

====================================================================

SECTION 13: BEST PRACTICES

====================================================================

- Always start with `#!/bin/bash`

- Quote variables: "$var"

- Check exit status: if [ $? -ne 0 ]; then ...

- Use functions to modularize

- Use logging and error redirection

- Avoid hardcoding sensitive info

- Validate inputs

====================================================================

SECTION 14: RESOURCES FOR FURTHER LEARNING

====================================================================

- Advanced Bash-Scripting Guide: https://tldp.org/LDP/abs/html/

- Bash manual: man bash

- ShellCheck (online linting): https://www.shellcheck.net/

- Learn Bash Scripting in Y Minutes: https://learnxinyminutes.com/docs/bash/

====================================================================

END OF GUIDE

====================================================================