1. Check if a file exists and is readable

```
#!/bin/bash
FILE="example.txt"
if [[ -r "$FILE" ]]; then
  echo "$FILE exists and is readable."
else
  echo "$FILE does not exist or is not readable."
fi
```

Explanation: -r checks if the file is readable. You can also use -e to check if it exists.

2. Pass arguments to a shell script

```
#!/bin/bash
echo "First argument: $1"
echo "Second argument: $2"
echo "All arguments: $@"
```

Explanation: $1, $2, etc., are positional parameters. $@ gives all arguments.

3. Count the number of lines in a file

```
#!/bin/bash
wc -l < "$1"
```

Explanation: wc -l counts lines. < avoids printing the filename.

4. Find and delete files older than 7 days

```
#!/bin/bash
find /path/to/dir -type f -mtime +7 -exec rm {} \;
```

Explanation: -mtime +7 finds files modified more than 7 days ago. -exec runs rm on each.

5. Difference between double and single quotes

```
name="Rajesh"
echo 'Hello $name'   # Outputs: Hello $name
echo "Hello $name"   # Outputs: Hello Rajesh
```

Explanation: single quotes are literal, double quotes allow variable expansion.

6. Monitor disk usage and alert if >90%

```
#!/bin/bash
THRESHOLD=90
USAGE=$(df / | grep / | awk '{print $5}' | sed 's/%//')
if [ "$USAGE" -gt "$THRESHOLD" ]; then
  echo "Disk usage is above $THRESHOLD%: $USAGE%"
fi
```

Explanation: df shows disk usage, awk and sed extract the percentage.

7. Handle errors and log them

```
#!/bin/bash
LOGFILE="script.log"
{
  echo "Starting script..."
  cp /nonexistent/file /tmp/
} >>"$LOGFILE" 2>&1
```

Explanation: 2>&1 redirects stderr to stdout, logging both to the file.

8. Parse log file for a keyword

```
#!/bin/bash
grep "ERROR" /var/log/syslog
```

Explanation: grep searches for lines containing "ERROR".

9. Use trap to clean up

```
#!/bin/bash
TMPFILE=$(mktemp)
trap "rm -f $TMPFILE" EXIT
echo "Temporary file: $TMPFILE"
# Do something with $TMPFILE
```

Explanation: trap ensures the temp file is deleted when the script exits.

10. Ping test for a list of servers

```
#!/bin/bash
for host in server1 server2 server3; do
  if ping -c 1 "$host" &>/dev/null; then
    echo "$host is reachable"
  else
    echo "$host is not reachable"
  fi
done
```

Explanation: ping -c 1 sends one ping. &>/dev/null suppresses output.

11. Automate user creation from a CSV file

```
#!/bin/bash
while IFS=',' read -r user pass; do
  useradd "$user"
  echo "$user:$pass" | chpasswd
done < users.csv
```

Explanation: Reads each line, creates a user, and sets the password using chpasswd.

12. Backup a directory and compress it with a timestamp

```
#!/bin/bash
SRC="/home/user/data"
DEST="/backup"
DATE=$(date +%Y%m%d_%H%M%S)
tar -czf "$DEST/backup_$DATE.tar.gz" "$SRC"
```

Explanation: Uses tar to compress the directory with a timestamped filename.

13. Menu-driven script for system tasks

```
#!/bin/bash
while true; do
  echo "1. Check uptime"
  echo "2. Check disk usage"
  echo "3. Check memory"
  echo "4. Exit"
  read -p "Choose an option: " choice
  case $choice in
    1) uptime ;;
    2) df -h ;;
    3) free -h ;;
    4) exit ;;
    *) echo "Invalid option" ;;
  esac
done
```

Explanation: Uses a loop and case to create a simple interactive menu.

14. Schedule a script using cron

crontab -e
Entry:
0 2 * * 1 /path/to/script.sh
Explanation: Runs the script every Monday at 2 AM. 0 2 * * 1 = minute, hour, day, month, weekday.

15. Compare two directories and list differences

```
#!/bin/bash
diff -qr /dir1 /dir2
```

Explanation: -q shows brief output, -r compares recursively.

16. Difference between exec, source, and running a script

exec script.sh: Replaces current shell with the script.
source script.sh or . script.sh: Runs in current shell, retains variables.
./script.sh: Runs in a new shell.
Example:

```
VAR="Hello"
source script.sh   # Can access VAR
./script.sh        # Cannot access VAR
```

17. Handle race conditions

```
#!/bin/bash
(
  flock -n 9 || exit 1
  echo "Running critical section"
) 9>/tmp/mylockfile
```

Explanation: flock prevents multiple instances from running simultaneously.

18. Redirect stderr to stdout

command 2>&1
Explanation: 2 is stderr, 1 is stdout. This merges both outputs.

19. Subshells and variable scope

```
#!/bin/bash
VAR="outside"
(
  VAR="inside"
  echo "Subshell: $VAR"
)
echo "Main shell: $VAR"
```

Output:
Subshell: inside
Main shell: outside
Explanation: Changes in subshell don·t affect parent shell variables.

20. Debug a complex script

```
#!/bin/bash
set -x  # Enable debugging
# Your script here
set +x  # Disable debugging
```

Explanation: set -x prints each command before executing it.