# Shell Scripting Solutions

My solutions to exercises in **Shell Scripting** by *Jason Cannon* (2015).



## Chapter 1. Shell Scripting, Succinctly

- Shebang - `#!`
- Variables
    - Assign a value to a variable - `VAR=value`
    - Assign command output to a variable - `VAR=$(command)`
    - Use a variable with `${VARIABLE}`
- Tests - `[ ]`
- The `if` statement - `if` `then` `fi`
- The `for` loop - `for` `in` `do` `done`
- Positional parameters - `$1` `$2` and `$@`
- Comments - `#`
- Getting user input - `read -p`

| TEST SYNTAX | DESCRIPTION |
| --- | --- |
| `[ -e FILE ]` | check if `FILE` exists |
| `[ -s FILE ]` | check if `FILE` exists and is not empty |
| `[ -f FILE ]` | check if `FILE` exists and is a regular file |
| `[ -d FILE ]` | check if `FILE` exists and is a directory |

| TEST SYNTAX | DESCRIPTION |
|---|---|
| `[ -r FILE ]` | check if `FILE` exists and is readable by user |
| `[ -w FILE ]` | check if `FILE` exists and is writable by user |
| `[ -x FILE ]` | check if `FILE` exists and is executable by user |
| `[ STR1 = STR2 ]` | check if strings `STR1` and `STR2` are equal |
| `[ STR1 != STR2 ]` | check if strings `STR1` and `STR2` are not equal |
| `[ -z STR ]` | check if string `STR` is empty |
| `[ -n STR ]` | check if string `STR` is nonempty |
| `[ STR ]` | check if string `STR` is nonempty |
| `[ -eq NUM1 NUM2 ]` | check if numbers `NUM1` and `NUM2` are equal |
| `[ -ne NUM1 NUM2 ]` | check if numbers `NUM1` and `NUM2` are not equal |
| `[ -lt NUM1 NUM2 ]` | check if numbers `NUM1` < `NUM2` |
| `[ -le NUM1 NUM2 ]` | check if numbers `NUM1` <= `NUM2` |
| `[ -gt NUM1 NUM2 ]` | check if numbers `NUM1` > `NUM2` |
| `[ -ge NUM1 NUM2 ]` | check if numbers `NUM1` >= `NUM2` |

## Exercise 1

Write a shell script that prints "Shell Scripting is Fun!" on the screen.

*Hint 1: Remember to make the shell script executable with the chmod command.*

*Hint 2: Remember to start your script with a shebang* `!`

**Solution:**

```
#!/bin/sh

echo "Shell Scripting is Fun!"
```

- `11.sh`

## Exercise 2

Modify the shell script from exercise 1 to include a variable. The variable will hold the contents of the message "Shell Scripting is Fun!".

**Solution:**

```
#!/bin/bash

MESSAGE="Shell Scripting is Fun!"
echo "$MESSAGE"
```

- `12.sh`

## Exercise 3

Store the output of the command "hostname" in a variable. Display "This script is running on ___." where "___" is the output of the "hostname" command.

*Hint: It's a best practice to use the* `${VARIABLE}` *syntax if there is text or characters that directly precede or follow the variable.*

**Solution:**

```
#!/bin/bash

HOSTNAME=$(hostname)
echo "This script is running on ${HOSTNAME}."
```

- `13.sh`

## Exercise 4

Write a shell script to check to see if the file `/etc/shadow` exists. If it does exist, display "Shadow passwords are enabled." Next, check to see if you can write to the file. If you can, display "You have permissions to edit /etc/shadow." If you cannot, display "You do NOT have permissions to edit /etc/shadow."

**Solution:**

```
```

```
#!/bin/bash

FILE="/etc/shadow"

if [ -e $FILE ]
then
    echo "Shadow passwords are enabled."
    if [ -w $FILE ]
    then
        echo "You have permissions to edit ${FILE}."
    else
        echo "You do NOT have permissions to edit ${FILE}."
    fi
fi
```

- `14.sh`

## Exercise 5

Write a shell script that displays "man", "bear", "pig", "dog", "cat", and "sheep" on the screen with each appearing on a separate line. Try to do this in as few lines as possible.

*Hint: Loops can be used to perform repetitive tasks.*

**Solution:**

```
#!/bin/bash

for ANIMAL in man bear pig dog cat sheep
do
    echo $ANIMAL
done
```

- `15.sh`

## Exercise 6

Write a shell script that prompts the user for a name of a file or directory and reports if it is a regular file, a directory, or another type of file. Also perform an `ls` command against the file or directory with the long listing option.

**Solution:**

```
#!/bin/bash

read -p "Please enter a name of a file or directory: " FILE

if [ -f $FILE ]
then
    echo "${FILE} is a regular file."
elif [ -d $FILE ]
then
    echo "${FILE} is a directory."
else
    echo "${FILE} is neither a regular file nor a directory."
fi

ls -l $FILE
```

- `16.sh`

## Exercise 7

Modify the previous script so that it accepts the file or directory name as an argument instead of prompting the user to enter it.

**Solution:**

```
#!/bin/bash

$FILE=$1

if [ -f $FILE ]
then
    echo "${FILE} is a regular file."
elif [ -d $FILE ]
then
```

```
    echo "${FILE} is a directory."
else
    echo "${FILE} is neither a regular file nor a directory."
fi

ls -l $FILE
```

- `17.sh`

## Exercise 8

Modify the previous script to accept an unlimited number of files and directories as arguments.

*Hint: You'll want to use a special variable.*

**Solution:**

```
#!/bin/bash

for FILE in $@
do
    if [ -f $FILE ]
    then
        echo "${FILE} is a regular file."
    elif [ -d $FILE ]
    then
        echo "${FILE} is a directory."
    else
        echo "${FILE} is neither a regular file nor a directory."
    fi
    ls -l $FILE
done
```

- `18.sh`

## Chapter 2. Exit Statuses And Return Codes

- Exit status / return code / exit code - an integer in `[0, 255]`
- Check exit status with `$?`
- Join two commands with `&&` and `||`
- Exit shell scripts with and without `exit`

### Exercise 1

Write a shell script that displays, "This script will exit with a 0 exit status." Be sure that the script does indeed exit with a `0` exit status.

**Solution:**

```
#!/bin/bash

echo "This script will exit with a 0 exit status."
exit 0
```

- `21.sh`
    - Check with `echo $?`

### Exercise 2

Write a shell script that accepts a file or directory name as an argument. Have the script report if it is a regular file, a directory, or another type of file. If it is a regular file, exit with a `0` exit status.  If it is a directory, exit with a `1` exit status.  If it is some other type of file, exit with a `2` exit status.

**Solution:**

```
#!/bin/bash

FILE=$1

if [ -f $FILE ]
then
    echo "${FILE} is a regular file."
    exit 0
elif [ -d $FILE ]
```

```
then
    echo "${FILE} is a directory."
    exit 1
else
    echo "${FILE} is neither a regular file nor a directory."
    exit 2
fi
```

- `22.sh`
    - Run with `./22.sh [FILENAME]` and check with `echo $?`

## Exercise 3

Write a script that executes the command `cat /etc/shadow`. If the command returns a `0` exit status, report "Command succeeded" and exit with a `0` exit status. If the command returns a non-zero exit status, report "Command failed" and exit with a `1` exit status.

**Solution:**

```
#!/bin/bash

cat /etc/shadow
if [ "$?" -eq "0" ]
then
    echo "Command succeeded"
    exit 0
else
    echo "Command failed"
    exit 1
fi
```

- `23.sh`
    - Run with `./23.sh` and `sudo ./23.sh`, respectively, and check with `echo $?`

## Chapter 3. Functions

- Define a function with `function func() {}`
- Call a function - no `()` needed
    - Define before use
- Positional parameters - `$1` `$2` and `$@` but no `$0`
- Variable scope
    - Global by default - blank if undefined
    - Local with `local`
- Exit statuses and return codes
    - Explicit - return from `0` to `255`
    - Implicit - exit status of last command in the function

### Exercise 1

Write a shell script that consists of a function that displays the number of files in the present working directory. Name this function `file_count` and call it in your script. If you use a variable in your function, remember to make it a local variable.

*Hint: The `wc` utility is used to count the number of lines, words, and bytes.*

**Solution:**

```
#!/bin/bash

function file_count() {
    local FILE_NUM=$(ls | wc -l)
    echo "$FILE_NUM"
}

file_count
```

- `31.sh`

## Exercise 2

Modify the script from the previous exercise. Make the `file_count` function accept a directory as an argument. Next, have the function display the name of the directory followed by a colon. Finally, display the number of files to the screen on the next line. Call the function three times. First on the "/etc" directory, next on the "/var" directory and finally on the "/usr/bin" directory.

Example output:

```
/etc:
    85
```

**Solution:**

```
#!/bin/bash

function file_count() {
    local DIR=$1
    local FILE_NUM=$(ls $1 | wc -l)
    echo "${DIR}:"
    echo "    $FILE_NUM"
}

file_count /etc
file_count /var
file_count /usr/bin
```

- `32.sh`
    - Sample Output

```
/etc:
    185
/var:
    13
/usr/bin:
    1419
```

## Chapter 4. Wildcards

- Two main wildcards - `*` `?`
- Character classes
    - e.g. `ca[nt]*` for "can", "cat", "candy", and "catch"
    - e.g. `[!aeiou]` for "be", but not "am", "is", or "are"
- Ranges
    - e.g. `[a-g]`
- Named character classes
    - `[:alpha:]`
    - `[:alnum:]`
    - `[:digit:]`
    - `[:lower:]`
    - `[:upper:]`
    - `[:space:]`

### Exercise 1

Write a shell script that renames all files in the current directory that end in `.jpg` to begin with today's date in the following format: YYYY-MM-DD. For example, if a picture of my cat was in the current directory and today was October 31, 2016 it would change name from `mycat.jpg` to `2016-10-31-mycat.jpg`.

*Hint: Look at the format options for the date command.*

*For **extra credit**, make sure to gracefully handle instances where there are no `.jpg` files in the current directory. (Hint: `man bash` and read the section on the `nullglob` option.)*

**Solution:**

```
#!/bin/bash

shopt -s nullglob

DATE=$(date +%F) # equivalently, date -I

for FILE in *.jpg
do
    mv $FILE ${DATE}-${FILE}
done
```

- `41.sh`

## Exercise 2

Write a script that renames files based on the file extension. The script should prompt the user for a file extension. Next, it should ask the user what prefix to prepend to the file name(s). By default, the prefix should be the current date in YYYY-MM-DD format. If the user simply presses enter, the current date will be used. Otherwise, whatever the user entered will be used as the prefix. Next, it should display the original file name and the new name of the file. Finally, it should rename the file.

- Example output 1:

```
Please enter a file extension: jpg
Please enter a file prefix:  (Press ENTER for 2015-08-10). vacation
Renaming mycat.jpg to vacation-mycat.jpg.
```

- Example output 2:

```
Please enter a file extension: jpg
Please enter a file prefix:  (Press ENTER for 2015-08-10).
Renaming mycat.jpg to 2015-08-10-mycat.jpg.
```

**Solution:**

```
#!/bin/bash

shopt -s nullglob

DATE=$(date +%F) # equivalently, date -I

read -p "Please enter a file extension: " EXT
read -p "Please enter a file prefix: (Press ENTER for ${DATE}). "
PREFIX

if [ -z $PREFIX ]
then
    PREFIX=${DATE}
fi

for FILE in *.${EXT}
```

```
do
    NEW_FILE=${PREFIX}-${FILE}
    echo "Renaming $FILE to ${NEW_FILE}."
    mv $FILE ${NEW_FILE}
done
```

- `42.sh`

## Chapter 5. Case Statements

- A typical case statement

```
read -p "Continue ([y]/n)? " ANSWER

case "$ANSWER" in
    [Yy]*)
        echo "Doing ..."
        ;;
    *)
        echo "Exit"
        ;;
esac
```

### Exercise 1

Create a startup script for an application called `sleep-walking-server`, which is provided below. The script should be named `sleep-walking` and accept `start` and `stop` as arguments. If anything other than `start` or `stop` is provided as an argument, display a usage statement: "Usage sleep-walking start|stop" and terminate the script with an exit status of `1`.

To start sleep-walking-server, use this command

- `/tmp/sleep-walking-server &`

To stop sleep-walking-server, use this command

- `kill $(cat /tmp/sleep-walking-server.pid)`

Here are the contents of `sleep-walking-server`. Be sure to put this file in `/tmp` and run `chmod 755 /tmp/sleep-walking-server` so that it is executable.

```
#!/bin/bash
# Instructions:
#   Place this script in /tmp
#
# Description:
#   This script simulates a service or a daemon.

PID_FILE="/tmp/sleep-walking-server.pid"
trap "rm $PID_FILE; exit" SIGHUP SIGINT SIGTERM
echo "$$" > $PID_FILE
while true
do
    :
done
```

**Solution:**

```
#!/bin/bash

case "$1" in
    "start")
        /tmp/sleep-walking-server &
        ;;
    "stop")
        kill $(cat /tmp/sleep-walking-server.pid)
        ;;
    *)
        echo "Usage: $0 start|stop"
        exit 1
        ;;
esac
```

- `sleep-walking-server`
- `51.sh`
    - Test

```
# Before Run
cp ./sleep-walking-server /tmp
chmod 755 /tmp/sleep-walking-server

# Expect "Usage: ./51.sh start|stop"
./51.sh help
```

```
# Expect Running
./51.sh start
[ -e /tmp/sleep-walking-server.pid ] && echo "Running"
```

```
# Expect Stopped
./51.sh stop
[ -e /tmp/sleep-walking-server.pid ] || echo "Stopped"
```
```

## Chapter 6. Logging

- `logger`

| OPTION | DESCRIPTION |
|---|---|
| `-p`, `--priority` | specify the priority with a pair of `facility.severity` e.g. `user.info` |
| `-t`, `--tag` | mark with specific tag |
| `-i` | log the PID |
| `-s`, `--stderr` | output the message to standard error as well |

*Start the logging service with e.g.* `sudo service rsyslog start` *first.*

### Exercise 1

Write a shell script that displays one random number on the screen and also generates a syslog message with that random number. Use the `user` facility and the `info` severity for your messages.

*Hint: Use* `$RANDOM`

**Solution:**
```

```
#!/bin/bash

MESSAGE="Random number: $RANDOM"

echo "$MESSAGE"

logger -p user.info "$MESSAGE"
```

- `61.sh`
    - *Run* `sudo service rsyslog start` *first, if the service is not yet started.*

### Exercise 2

Modify the previous script so that it uses a logging function. Additionally, tag each syslog message with `randomly` and include the process ID. Generate 3 random numbers.

**Solution:**

```
#!/bin/bash

function logit() {
    local MESSAGE=$@
    echo "$MESSAGE"
    logger -p user.info -t "randomly" -i "$MESSAGE"
}

logit "Random number: $RANDOM"
logit "Random number: $RANDOM"
logit "Random number: $RANDOM"
```

- `62.sh`

## Chapter 7. Debugging

- Common debugging techniques
    - `#!/bin/bash -x`
    - `set -x` - print commands with variables and wildcards expanded
    - `set -e` - stop immediately if a command exits with a non-zero status

        - `set -v` - print commands as they are, i.e. no expansion
        - `DEBUG=true; if $DEBUG; then ...`
        - `DEBUG=true; $DEBUG && echo "Debug mode ON"`
        - `DEBUG=false; $DEBUG || echo "Skipping a time-consuming command ..."`
        - `DEBUG="echo"; $DEBUG ls`
        - Use user-defined `debug()` function

```
#!/bin/bash

function debug() {
    echo "Executing: $@"
    $@
}
```

        - `PS4='+ ${BASH_SOURCE}:${LINENO}:${FUNCNAME[0]}():'`
- Text file errors such as `^M` when working across operating systems
    - Check line endings
        - Unix-style - `LF` - linefeed `\n`
        - Windows-style - `CRLF` - carriage return and linefeed `\r\n`
    - Use `cat -v`
    - Use `dos2unix` and `unix2dos`

## Exercise 1

Write a shell script that exits on error and displays commands as they will execute, including all expansions and substitutions. Use 3 `ls` commands in your script. Make the first one succeed, the second one fail, and the third one succeed. If you are using the proper options, the third `ls` command will not be executed.

**Solution:**

```
#!/bin/bash -ex

ls .
ls /not/here
# should not be here
ls .
```

- `71.sh`

### Exercise 2

Modify the previous exercise so that script continues, even if an error occurs. This time, all three `ls` commands will execute.

**Solution:**

```
#!/bin/bash -x

ls .
ls /not/here
ls .
```

- `72.sh`

## Appendix I. Shell Script Checklist

1. Start with shebang?

```
#!/bin/bash
```

2. Then a comment for scirpt purpose?

```
# This script creates backups.
```

3. Then global variables declared following the comment?

```
DEBUG=true
```

4. Then functions grouped after global variables?

```
function debug() {
    echo $@
    $@
}
```

5. Keyword `local` for local variables in functions?

```
fucntion greet() {
    local MESSAGE="Hello, World!"
    echo "$MESSAGE"
}
```

6. Then the main body of the script?

```
echo "Start"
```

7. Script exit with an explicit exit status?

```
exit 0
```

8. Exit status explicitly used at all the exit points?

```
if [ ! -d "$DIR" ]
then
    echo "$DIR does not exist. Exit."
    exit 1
fi
```

## Appendix II. Shell Script Template

```
#!/bin/bash

# This script greets users.

GLOBAL_VAR="World"

function greet() {
    local LOCAL_VAR=1
    local NAME_LIST=$@

    if [ $# -eq 0 ]
    then
        echo "Hello, ${GLOBAL_VAR}!"
    else
        for NAME in $NAME_LIST
        do
            echo "${LOCAL_VAR}: Hello, ${NAME}!"
            LOCAL_VAR=$((${LOCAL_VAR}+1))
```

```
        done
    fi
}

greet $@

exit 0
```

- `template.sh`
    - Sample Output

```
$ ./template.sh
Hello, World!

$ ./template.sh Aaron Bobby Charlie
1: Hello, Aaron!
2: Hello, Bobby!
3: Hello, Charlie!
```