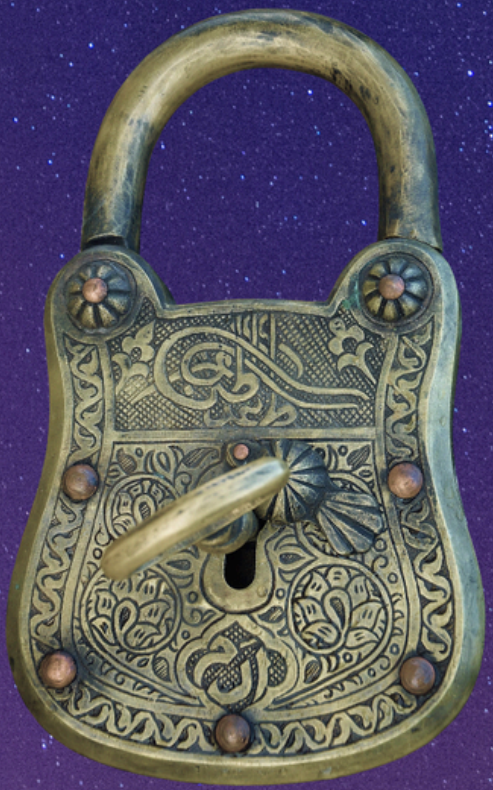
DECIPHER BASH SCRIPTING

AN EASY TO FOLLOW TECHNIQUE THAT WILL HELP YOU READ BASH SCRIPTS FAST



ABOUT CODEFATHERTECH

CodeFatherTech is committed to transforming the knowledge of aspiring developers worldwide. Our mission is to guide professionals in their growth and support them in fulfilling their potential.

WE WANT TO HELP YOU

Get Started With Programming Fast

Become Confident About Your Programming Skills

Transform Your IT Career

Be the Go-to Person in Your Working Environment

CODEFATHERTECH

hello@codefather.tech

How easy is for you to read a Bash script that someone else has written? Are you fed up of having to work with Unix or Linux and not being comfortable with Bash scripting?

That's okay, this is the right place where to start. Use this checklist as a step-by-step process that will help you read Bash scripts from scratch and will make your life easier.

This checklist provides a 7-step approach to go through any script...I have created this as a result of years of experience. Apply the steps in this checklist and you will have an easy framework to quickly identify the building blocks of Bash scripts (and Shell scripts in general).

Let's get started!

Claudio Sabato

STEP 1

CLARIFY THE CONTEXT

Let's say you are reading a Bash script for the first time...

It can be confusing if you don't know what to look for.

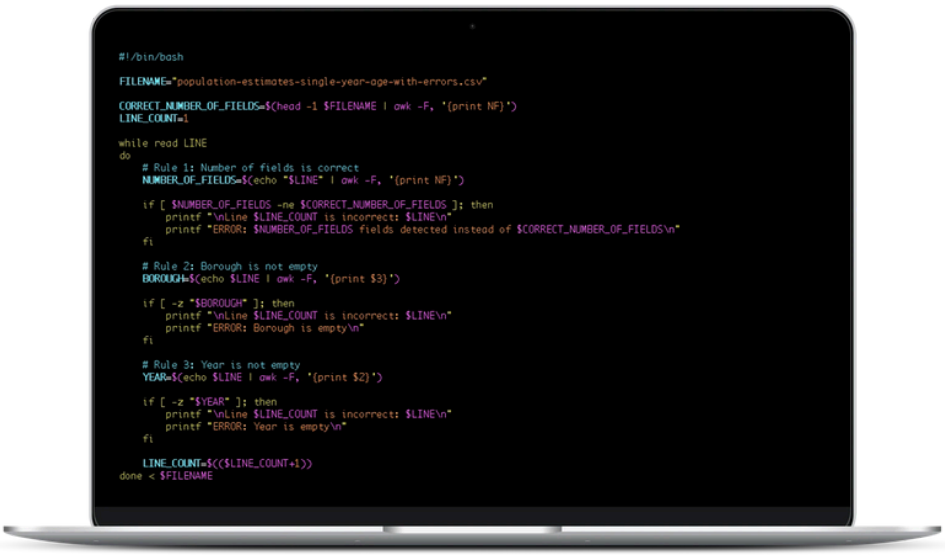
In this first step, we will start looking at the context in which the script has been written, and in the next steps, you will learn different things you can look for in the script to "decipher" it.

So, going back to the context...

Is it clear to you why the script was written?

What is it supposed to do?

This is really important before delving into the code.



It's harder to read code without knowing why it was created, so before focusing on the code, find as much as you can about it:

1. Is the person who created the script still around?
2. Has anyone used or updated the script apart from the person who wrote it?
3. What problem is the script solving?
4. How is the script solving that problem?
5. Is there any documentation about the way the script is structured?
6. Has anyone documented how to run the script?

Answering these questions can make a big difference in helping you understand the script.

And I know it's not always possible to find someone who knows about the script or some documentation about it, especially if it was written a long time ago. But it's worth giving it a try.

And now that you have more details about the script, we can start looking at the code!

STEP 2

GO THROUGH ALL THE COMMENTS

A good programming practice is to add comments to your code. Comments do two things:

1. Help developers go back to scripts they have written after a long time (if you go back to a script you wrote one year before, you may even doubt you were the one who wrote it!)
2. Help those who haven't written the script understand it without driving them crazy trying to figure out what was in the developer's head when they were writing the script.

You can identify comments by looking at lines that start with the # character.

```
# This is a comment
function validate_field {
    FIELD_INDEX=$1
    HEADER=$2
```

Most editors display comments with a different colour to distinguish them from the rest of the code.

At the beginning, go through all the comments in the script without worrying about the rest of the code. This will help you build a better understanding of the context (see STEP 1) and it will give you a more specific knowledge about the code.

STEP 3

FIGURE OUT HOW TO RUN THE SCRIPT

Being able to execute the script is critical for you to understand how the script works.

There are two options here:

1. The script doesn't require any arguments when it's executed.
2. The script requires one or more arguments when it's executed.

How do you figure out if you are in the scenario 1 or 2? There are a few ways...

Documentation

If the script is documented, look for examples of how to execute it to see if it expects any arguments or if it doesn't.

Help included in the script

Search in the script for a "help" section. Adding a help section to a script is a standard approach to show you:

- How to use the script?
- How many arguments does it accept?
- What is the meaning of each argument?

The common standard is to print the help of the script when the script is executed, passing flags like **-h** or **--help**.

For example, if you have a script called run_me.sh here is how you would show its help (if it has been implemented by the developer):

run_me.sh -h or **run_me.sh --help**

Arguments handled in the code of the script
Look for the variables $1, $2, $3, etc, in the script, where $1 is the variable that contains the first argument passed to the script, $2 the second argument, and so on.

Here is a basic check you can use:

- If you don't see $1, $2, $3, etc, in the script, the script doesn't require any arguments.
- If you see $1, $2, $3, etc, in the script, the script requires one or more arguments.

Here is how you can execute a Bash script called run_me.sh that doesn't require any arguments:

run_me.sh

and here is how the command changes if the same script requires two arguments:

run_me.sh arg1 arg2

Other programs executing the script
Check if there are other programs that execute the script.

One example could be a job scheduling program that executes the script on a specific schedule (e.g. every day at midnight).

Looking at how the script is executed by this program can help you confirm if it requires any arguments or not.

Before moving on...

I know I have mentioned variables a few times now, let's clarify what a variable is (we will also cover it in the next step):

A variable is used to store information required during the execution of a script.

For example, the value of the variable SEASON could be:

- “Autumn”
- “Winter”
- “Spring”
- “Summer"

STEP 4

BUILD A LIST OF ALL THE VARIABLES

Variables are used to store temporary information required in the execution of a script.

For example, if you create a script that shows all the types of animals in a zoo, you could have a few variables to represent different concepts in your script.

Examples of variables could be:

- ZOO
- ANIMAL
- NUMBER_OF_ANIMALS

A process that will help you understand how variables are used is the following:

1. Go through every variable in the script.
2. See if you can understand what that variable is used for.

This could be straightforward if the name of a variable is self-explanatory; unfortunately, this depends on how well the script is written.

Not every developer takes the time to give variables appropriate names...

Make sure you do! :)

If the name of a variable is not clear, you will have to take more time and try to learn about that variable by looking at how it's used in the script.

A value is assigned to a variable using the following syntax (e.g., for a variable of type string or in other words a variable that holds a sequence of characters):

VARIABLE_NAME="value"

To find variables quickly, look for = signs in the script; the variable name is on the left side of the equal sign.

Now, let's say you can see that the variable ANIMAL is defined in the script.

Have you noticed that sometimes you see just ANIMAL in the script and sometimes $ANIMAL?

The reason behind that is the following:

- ANIMAL is used when assigning a value to the variable.
- $ANIMAL is used to read the value of that variable.

STEP 5

LOCATE FUNCTIONS

A function is a sequence of commands that can be executed multiple times to perform a very specific task.

Here are three examples of what a function could be used to:

1. Download a file from the Internet to a specific directory.
2. Delete files older than 7 days from your computer.
3. Verify the value of a variable matches a set of expected values.

In reality, the options of what a function could do are pretty much endless!

So, how can you identify functions in a script? Look for one of the following constructs:

```
name_of_the_function() {
   <list_of_commands>
}
```

or

```
function name_of_the_function {
   <list_of_commands>
}
```

For example, if the name of the function is *download_file* you will see:

```
download_file() {
   <list_of_commands>
}
```

or

```
function download_file {
   <list_of_commands>
}
```

As you can see in both cases, the list of commands in a function is enclosed between curly brackets {}.

Identify every function in your script and try to understand what the purpose of that function could be.

Naming standards

Using clear names for functions makes the code a lot more readable. Ideally, the name of a function should be everything you need to understand what that function does.

If you have to go through each line in a function to understand what the function does, that's an indication that the name of the function is not clear enough.

Below you can see an example of a function:

```
function verify_status_code {
    STATUS_CODE=$1
    OPERATION=$2

    if [ $STATUS_CODE -eq 0 ]; then
        printf "\n$OPERATION successful\n"
    else
        printf "\n$OPERATION failed\n"
        exit
    fi
}
```

STEP 6

RECOGNISE COMMON STATEMENTS

Two constructs that are very common in a script (this applies to coding in general) are:

- conditional statements
- loops.

Conditional statements

Conditional statements are used to make decisions in your script.

Imagine you reach a crossroad, you have to decide to go left or right, and you choose left because you are going back home.

That's the same thing that happens in a conditional statement...

Here is an example:

```
if [ "destination" = "home" ]; then
    <list_of_commands_to_go_left>
else
   <list_of_commands_to_go_right>
fi
```

Loops

Loops are used to execute the same sequence of commands as long as a specific condition is true:

```
for INDEX in `seq 1 5`;
do
   echo $INDEX
done
```

The code above prints the numbers from 1 to 5.

The echo command is used to print messages in the shell and **$INDEX** refers to the value of the INDEX variable (as explained at STEP 4).

Identify conditional statements and loops in your script and see how this adds another dimension to your knowledge of the script.

STEP 7

IDENTIFY THE BUILDING BLOCKS

Now you know the different elements you can identify in a script. It's time to use this knowledge!

Try to split the script into logical blocks where each block performs a specific task.

Don't forget to use your knowledge about:

- Context
- Comments
- How to run the script
- Variables
- Functions
- Conditional statements and loops

In most scripts, you will be able to identify a list of commands that, when executed in sequence, lead to the expected outcome.

For example, a script that generates a report could be doing the following:

1. Connect to a database
2. Retrieve data for the last 7 days
3. Filter the data
4. Export the filtered data into CSV format

As you can see, we have a sequence of four logically distinct tasks that, when executed in the right sequence, generate the report.

Do your best to identify the logical blocks in your script; this will give you a deeper understanding of how the script works.

And don't worry if this isn't easy at the beginning...

It's perfectly normal, and it will become easier with practice.

BONUS STEP

LOOK FOR OS COMMANDS

One of the reasons why Bash scripting is very powerful is the possibility of including **Operating System (OS) commands** (e.g. find, awk, grep, etc...) in your scripts.

Make a list of all the Operating System commands you can see in the script you are analysing and become familiar with any commands you are not already familiar with.

Don't forget to use the **man** command if it's not clear to you how a command is used in the script you are reading.

For example, if you want to know more about the wc command, use:

man wc

This is a very good way to improve your Operating System knowledge while developing your scripting skills.

And now it's time for you to go through a real script and apply what you have learned here.

TEST YOUR KNOWLEDGE

Practice what you have learned in this guide using the script below:

```
#!/bin/bash

function verify_website {
    if [ $# -ne 1 ]; then
        echo "Please specify the website"
        exit
    fi
}

# What do you think $1 is?
WEBSITE=$1

verify_argument $WEBSITE

# Check if the website is reachable
ping -c 4 $WEBSITE

# What's the difference between IP_ADDRESS and $IP_ADDRESS?
IP_ADDRESS=`dig +short $WEBSITE`
echo "The IP address for $WEBSITE is $IP_ADDRESS"

MESSAGE="Script execution completed"
echo $MESSAGE
```

Is it clear to you what this script does?

Can you see any comments, functions, variables or Operating System commands?

There is an error in the code, can you see where?
HINT: It's related to the function at the beginning of the script.

WHAT'S NEXT?

You now have a standard approach to read a Bash script for the first time and understand why it has been written, how it can be executed, and what logical blocks it is made of.

Now it's time for you to select 2-3 scripts you have been struggling with and go through each one of the 7 steps for every script. This is the way to get the most out of this checklist.

If you have any questions or if you need any support feel free to email us at **hello@codefather.tech**.

To your success!

Claudio Sabato