

# Workshop 05 - Dot Files, Bash Scripting and Cronjobs

---

## Why Scripting?

There are certain actions that we can do in the terminal that could and should be automatically triggered, be it by the time of the day or by an external process. This could be anywhere from a database backup to a daily report of tables in a database.

## How This Works

We can test how a script works by creating one in our host machine, inside of our database folder using:

```
touch hello.sh
chmod +x hello.sh
code hello.sh
```

Touch creates the file and chmod gives it execution rights.

Code is only to open it using Visual Studio Code, but we can do it manually or edit it using other means.

### Inside our sh file

Every bash file starts with `#!/bin/bash`, this is just to indicate to the interpreter the language we're using.

To test it, we can use a simple `echo "Hello World!"` and try to run it.

### Actually Running It

Now, running a file in the `.` directory is not as easy as it seems, for we can only run files that are explicitly stated in the `$PATH` variable as a safety measure.

So, to run the program, we specify the directory: `./hello.sh`. This might behave differently in Windows, in my case I had to specify `bash hello.sh` for it to work, this is temporal however, for in other files the credentials are successfully changed when I did it from the host machine.

```
vagrant@database:/vagrant$ bash hello.sh
Hello world!
vagrant@database:/vagrant$ |
```

## Continue With the Script

There are tons of things we can do with a bash script, for now, we can use a simple if.

```
#!/bin/bash

echo "Please type your name"
read name
```

```
if [ -z "$name" ]; then
    echo "I would've loved to meet you"
else
    echo "Hello $name"
fi
```

```
vagrant@database:/vagrant$ bash hello.sh
Please type your name
Pamela
Hello Pamela
vagrant@database:/vagrant$ bash hello.sh
Please type your name

I would've loved to meet you
vagrant@database:/vagrant$
```

With another file, we can tell the user their generation according to their birth year.

```
#!/bin/bash

echo "Please type your birth year:"
read year

if [[ -z "$year" ]]; then
    echo "You didn't type anything"
elif [[ ! "$year" =~ ^[0-9]{4}$ ]]; then
    echo "You should introduce an actual number"
elif [[ "$year" -lt 1900 && "$year" -gt 2025 ]]; then
    echo "The year must be between 1900 and 2024"
else
    if [ "$year" -ge 1946 ] && [ "$year" -le 1964 ]; then
        echo "You're a Baby Boomer"
    elif [ "$year" -ge 1965 ] && [ "$year" -le 1980 ]; then
        echo "You're from the X Generation"
    elif [ "$year" -ge 1981 ] && [ "$year" -le 1996 ]; then
        echo "You're a Millenial"
    elif [ "$year" -ge 1997 ] && [ "$year" -le 2012 ]; then
        echo "You're from the Z Gen"
    elif [ "$year" -ge 2013 ]; then
        echo "You're from the Alpha Generation"
    else
        echo "We don't have a generation for you"
    fi
fi
```

```
vagrant@database:/vagrant$ bash generation.sh
Please type your birth year:
2004
You're from the Z Gen
vagrant@database:/vagrant$
```

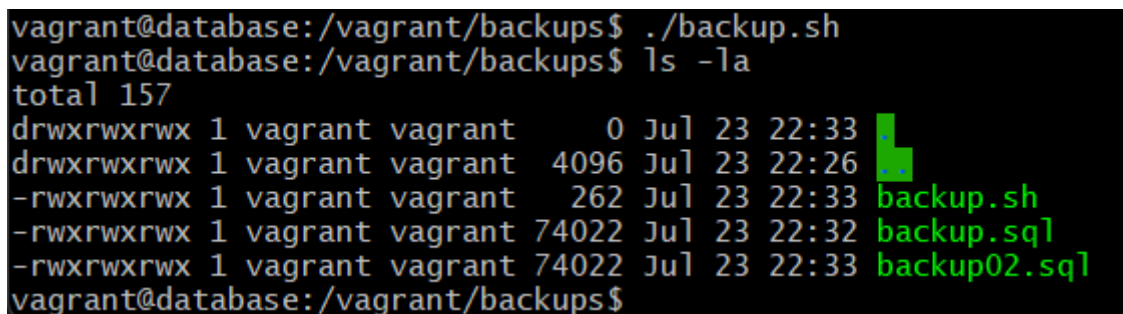
## Database Backups

To further explain one of the most important jobs of the scripts, we should do a basic backup. To do this, we'll create a new folder dedicated to it and the corresponding file inside of it:

```
mkdir backups\  
cd backups  
touch backup.sh  
code backup.sh
```

Inside of our file, we'll run the following code:

```
#!/bin/bash  
  
database="northwind"  
username="north"  
password="secret" # In a professional environment this should be obtained through  
a .env variable  
directory="/vagrant/backups/"  
  
cd $directory  
mysqldump -u $username --password=$password $database > backup02.sql
```



```
vagrant@database:/vagrant/backups$ ./backup.sh  
vagrant@database:/vagrant/backups$ ls -la  
total 157  
drwxrwxrwx 1 vagrant vagrant    0 Jul 23 22:33 .  
drwxrwxrwx 1 vagrant vagrant 4096 Jul 23 22:26 ..  
-rwxrwxrwx 1 vagrant vagrant  262 Jul 23 22:33 backup.sh  
-rwxrwxrwx 1 vagrant vagrant 74022 Jul 23 22:32 backup.sql  
-rwxrwxrwx 1 vagrant vagrant 74022 Jul 23 22:33 backup02.sql  
vagrant@database:/vagrant/backups$
```

This will create, unsurprisingly, a backup of the database northwind. A timestamp can also be saved by adding it to the variables: `timestamp=$(date +"%Y%m%d%H%M%S")` and introducing it as the name: `mysqldump -u $username --password=$password $database > ${database}_${timestamp}.sql`

### Hashing the Password

As said before in the script comments, we should not have our password out and about in the file, we should have it in our machines and our machines only.

To establish an environment password, we simply 'export' it: `export dbpassword=secret` and now we can use it in our script.

```
if [ -z "$dbpassword" ]; then  
    echo "Error: dbpassword does not exist"
```

```
    exit 1
fi
```

Make sure to change 'password' to 'dbpassword' in the command.

```
vagrant@database:/vagrant/backups$ ./backup.sh
vagrant@database:/vagrant/backups$ ls -la
total 81
drwxrwxrwx 1 vagrant vagrant    0 Jul 23 22:46 .
drwxrwxrwx 1 vagrant vagrant 4096 Jul 23 22:26 ..
-rwxrwxrwx 1 vagrant vagrant  306 Jul 23 22:46 backup.sh
-rwxrwxrwx 1 vagrant vagrant    0 Jul 23 22:46 northwind_20240723224619.sql
-rwxrwxrwx 1 vagrant vagrant 74022 Jul 23 22:46 northwind_20240723224644.sql
vagrant@database:/vagrant/backups$
```

## Accepting more Arguments

We can add things to the file while running the command to run the file in our machines:

```
#!/bin/bash

database="northwind"
username="north"
directory="/vagrant/backups/"
timestamp=$(date +"%Y%m%d%H%M%S")
filename=$1

# Validate dbpassword configuration
if [ -z "$dbpassword" ]; then
    echo "Error: dbpassword does not exist"
    exit 1
fi

# Validate new argument
if [ -z "$1" ]; then
    filename=${database}_${timestamp}.sql
else
    filename=${database}_${filename}.sql
fi

cd $directory
mysqldump -u $username --password=$dbpassword $database > $filename
```

Here, we accept an argument for it to be the filename and we change it depending on whether or not the user introduced additional text while running the command.

```
vagrant@database:/vagrant/backups$ ./backup.sh
vagrant@database:/vagrant/backups$ ./backup.sh tuesday
vagrant@database:/vagrant/backups$ ls -la
total 157
drwxrwxrwx 1 vagrant vagrant 0 Jul 23 22:51 .
drwxrwxrwx 1 vagrant vagrant 4096 Jul 23 22:26 ..
-rwxrwxrwx 1 vagrant vagrant 469 Jul 23 22:51 backup.sh
-rwxrwxrwx 1 vagrant vagrant 74022 Jul 23 22:51 northwind_20240723225132.sql
-rwxrwxrwx 1 vagrant vagrant 74022 Jul 23 22:51 northwind_tuesday.sql
vagrant@database:/vagrant/backups$
```

Now, even though this database is very small, it is convenient for us to compress each and every backup we make or things could get heavy pretty easily. To do this, we can compress the original file and then remove it:

```
.
.
.

cd $directory
mysqldump -u $username --password=$dbpassword $database > $filename.sql

tar vcfz $filename.tar.gz $filename

rm $filename.sql
```

## Registering logs

Now, this is all important information and we should keep track of it using the very famous logs. To do this, we have been introduced to the concept of a function in bash this way:

```
log_message() {
    local MESSAGE=$1
    echo "$timestamp : $MESSAGE" >> $logfile
}
```

And then to call it we use `log_message "Message"` in all the places we consider them necessary.

```
vagrant@database:/vagrant/backups$ rm backup.log
vagrant@database:/vagrant/backups$ rm *.tar.gz
vagrant@database:/vagrant/backups$ ./backup.sh
northwind_20240723230925.sql
vagrant@database:/vagrant/backups$ cat backup.log
20240723230925 : Starting northwind backup
20240723230925 : northwind backup successfully completed on northwind_20240723230925
sql
20240723230925 : Compression successfully completed
vagrant@database:/vagrant/backups$
```

Additionally, we can print this logs on screen using a tee pipe:

```
log_message() {
    local MESSAGE=$1
    echo "$timestamp : $MESSAGE" | tee -a $logfile
}
```

## Executing this Automatically

To make a cronjob that does that when we want we go to `crontab -e` to see the file and add ,uncommented, of course, `51 23 23 07 2 dbpassword=secret /vagrant/backups/backup.sh tuesday23` this will add a new backup at 11:51 pm on the tuesday 23 of july. Furthermore, we can ask for a log on this adding after 'tuesday23' >> `/vagrant/backups/cron.sql 2>&1`

```
vagrant@database:/vagrant/backups$ date
Tue Jul 23 23:50:54 UTC 2024
vagrant@database:/vagrant/backups$ ls -la
total 29
drwxrwxrwx 1 vagrant vagrant 4096 Jul 23 23:09 .
drwxrwxrwx 1 vagrant vagrant 4096 Jul 23 22:26 ..
-rwxrwxrwx 1 vagrant vagrant 184 Jul 23 23:09 backup.log
-rwxrwxrwx 1 vagrant vagrant 971 Jul 23 23:32 backup.sh
-rwxrwxrwx 1 vagrant vagrant 12842 Jul 23 23:09 northwind_20240723230925.sql.tar.gz
vagrant@database:/vagrant/backups$ date
Tue Jul 23 23:51:00 UTC 2024
vagrant@database:/vagrant/backups$ ls -la
total 45
drwxrwxrwx 1 vagrant vagrant 4096 Jul 23 23:51 .
drwxrwxrwx 1 vagrant vagrant 4096 Jul 23 22:26 ..
-rwxrwxrwx 1 vagrant vagrant 445 Jul 23 23:51 backup.log
-rwxrwxrwx 1 vagrant vagrant 971 Jul 23 23:32 backup.sh
-rwxrwxrwx 1 vagrant vagrant 12842 Jul 23 23:09 northwind_20240723230925.sql.tar.gz
-rwxrwxrwx 1 vagrant vagrant 12839 Jul 23 23:51 northwind_tuesday23.sql.tar.gz
vagrant@database:/vagrant/backups$
```

```
vagrant@database:/vagrant/backups$ ls -la
total 29
drwxrwxrwx 1 vagrant vagrant 4096 Jul 23 23:59 .
drwxrwxrwx 1 vagrant vagrant 4096 Jul 23 22:26 ..
-rwxrwxrwx 1 vagrant vagrant 261 Jul 23 23:59 backup.log
-rwxrwxrwx 1 vagrant vagrant 971 Jul 23 23:32 backup.sh
-rwxrwxrwx 1 vagrant vagrant 285 Jul 23 23:59 cron.sql
-rwxrwxrwx 1 vagrant vagrant 12839 Jul 23 23:59 northwind_tuesday23.sql.tar.gz
vagrant@database:/vagrant/backups$ cat cron.sql
20240723235901 : Starting northwind backup
20240723235901 : northwind backup successfully completed on northwind_tuesday23.sql
20240723235901 : Starting compression
northwind_tuesday23.sql
20240723235901 : Removing uncompressed file
20240723235901 : Compression successfully completed
vagrant@database:/vagrant/backups$
```

The date at which the backup is made can and should vary depending on the requirements that need to be met.

```
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow   command
0 3 * * 1 dbpassword=secret /vagrant/backups/backup.sh >> /vagrant/backups/cron.sql 2>&1
* * * * * dbpassword=secret /vagrant/backups/backups.sh >> /vagrant/backups/cron.sql 2>&1
# 0 sunday
# 1 monday
# 2 tuesday
# 3 wednesday
# 4 thursday
# 5 friday
```

## Configuring an alias

To change an alias, we can go to the `.bashrc` file. We can do this by going to the root of our file system using `~`, and then `cd .bashrc`, opening it with nano we can add an alias `alias ll='ls -l'` to make our work faster.

```
vagrant@database:/$ ll
total 64
lrwxrwxrwx   1 root    root      7 May  3 08:34 bin -> usr/bin
drwxr-xr-x   3 root    root    4096 May  3 08:35 boot
drwxr-xr-x  16 root    root   3180 Jul 24 00:31 dev
drwxr-xr-x  67 root    root   4096 Jul 24 00:31 etc
drwxr-xr-x   3 root    root   4096 May  3 08:35 home
lrwxrwxrwx   1 root    root      7 May  3 08:34 lib -> usr/lib
lrwxrwxrwx   1 root    root      9 May  3 08:34 lib64 -> usr/lib64
drwx-----  2 root    root  16384 May  3 08:34 lost+found
drwxr-xr-x   2 root    root   4096 May  3 08:34 media
drwxr-xr-x   2 root    root   4096 May  3 08:34 mnt
drwxr-xr-x   2 root    root   4096 May  3 08:34 opt
dr-xr-xr-x 138 root    root      0 Jul 24 00:31 proc
drwx-----  4 root    root   4096 Jul  9 00:39 root
drwxr-xr-x  19 root    root    560 Jul 24 00:49 run
lrwxrwxrwx   1 root    root      8 May  3 08:34/sbin -> usr/sbin
drwxr-xr-x   2 root    root   4096 May  3 08:34 srv
dr-xr-xr-x  13 root    root      0 Jul 24 00:31 sys
drwxrwxrwt   8 root    root   4096 Jul 24 00:31 tmp
drwxr-xr-x  12 root    root   4096 May  3 08:34 usr
drwxrwxrwx   1 vagrant vagrant 4096 Jul 23 22:26 vagrant
drwxr-xr-x  11 root    root   4096 May  3 08:34 var
```

---

Created by Pamela Murillo