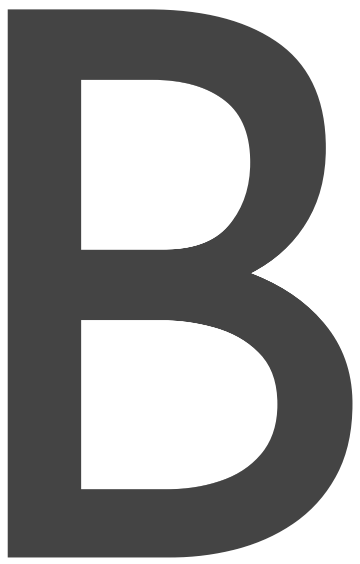
codequoi

- HOME
- CATEGORIES
    - C PROGRAMMING
    - COMPUTER SCIENCE
    - 42 SCHOOL PROJECTS
    - PROGRAMMING TOOLS
- ABOUT
- CONTACT
-
- Type here to search... SEARCH
-

Born2beroot 02: Configuring a Debian Virtual Server

By Mia Combeau
In 42 School Projects
March 11, 2022
24 Min read
Add comment



Having previously installed a virtual machine for the 42 project Born2beroot, we must now set up our new server. In particular, the subject requires the configuration of security measures by implementing sudo, a firewall, a strong password policy, and a script called `monitoring.sh`. In addition, we will learn to manage users and the groups to which they belong.

Our virtual server this article series runs Debian 11.2 (bullseye) on VirtualBox 6.1.

Born2beroot : Installation | **Configuration** | Bonus | Subject [pdf]

Table of Contents

Useful Commands for Born2beroot

The basic commands we have to know and which will no doubt be very useful during the Born2beroot server's configuration are:

- **`exit`** or **`logout`**: quits the current session and returns us to the connection prompt.
- **`su other_username`**: connect as a different user. We will be asked for the other user's password. Running the **`su`** (*switch user*) command without specifying a username, will let us log in as the default user, root. We can **`exit`** at any time to return to the previous user session.
- **`systemctl reboot`**: restarts the system (requires root permissions)
- **`systemctl poweroff`**: turns off the system (requires root permissions)

Packet Management in Debian: Apt and Aptitude

In Linux-based operating systems like Debian, installing software is done through packets. These packets contain all the files necessary to the implementation of a set of commands or functionalities. Therefore, we need to understand how to install these packets on our Born2beroot virtual server.

APT

APT (*Advanced Packaging Tool*) is a program that fetches packages from official sources in order to install, update or delete them along with their dependencies. It can only be used vie the command line. Originally created for

Debian, APT now comes preinstalled with the majority of Linus distributions.

APT is in reality a collection of several tools, in particular: **`apt-get`**, **`apt-cache`** and **`apt-config`**. To install a packet, we can write `apt install <package_name>`, which is exactly the same as `apt-get install <package_name>`. Similarly, to view the details of a packet, we can do `apt search <package_name>` or `apt-cache search <package_name>`.

Aptitude

Aptitude is a very similar tool with the same goal, but it is a higher-level packet manager. It integrates the APT's functionalities and offers a graphical interface on top of the text-based interface. The latter is also more informative than APT: for instance, it annotates which packets are actually installed or not on the system while performing a search. Aptitude is also more effective at conflict resolution during installations and deletions, and we can ask it why a certain package is recommended with the commands `why` and `why-not`.

To install Aptitude, all we need to do is **`apt install aptitude`**. Root permissions are required for installations, so read the following section about root and sudo in order to be able to install Aptitude. For the rest of this article, we will continue using APT, but there are corresponding commands for Aptitude.

Packet Updates and Upgrades

Before installing a new packet, we should always update the manager's packet list to the latest version. It's also a good idea to upgrade the packets themselves at the same time. These actions require root permissions.

```
apt update    // updates the list of available pac
apt upgrade   // updates the packages themselves
```

Note: if the update fails with an error message like “repository is not valid yet”, it might be because the virtual machine was restored from a snapshot and is out of sync time-wise. Check the system’s time and date with the `timedatectl` command, it might be in the past! If that is the case, all we need to do is reboot the system and wait a minute for the clock to sync.

Root, Superuser and Sudo

A regular user on our Born2beroot server cannot install packets, or even upgrade them. For that, we need to access the super-user, root. We configured it during Debian’s installation and, with a little luck, we still remember its password!

Root is truly the most powerful user, the one at the root of a Debian installation. It can do absolutely anything on the machine. And, as they say, “with great power comes with great responsibility”. It is far better to never connect as root, where a small mistake can have big consequences. But we still need root privileges from time to time…

sudo (*super-user do*) is a program that allows certain users to execute certain commands as root, from their own session. That way, users don’t even need to know the root password: `sudo` asks for their own password instead. Each `sudo` command is also logged as a security measure.

Installing Sudo on Born2beroot

In order to install sudo, we need to exceptionally log in as root.

```
$ su root
```

It will ask for the root password. Once logged in, we may notice that the command prompt sign $ changed to a # to mark that we now have root access.

```
# apt update
# apt upgrade
# apt install sudo
# sudo --version
```

Now, we need to give our regular user the ability to use sudo, by adding the user to the sudo group (and checking that it was added successfully):

```
# sudo usermod -aG sudo <username>
# getent group sudo
```

To check if we now have sudo privileges, we can `exit` the root session to go back to our own user, then run the following command:

```
$ sudo whoami
```

Once our password entered, we should get, as an answer, “root”. If that doesn’t work, we might have to log out and log back in for the changes to take effect.

As a very last resort, if the user still has no sudo privileges after logging out and back in again, we will have to modify the `sudoers.tmp` file from the root session with the following commands:

```
$ su
# sudo visudo
```

And add this line to the file:

```
username   ALL=(ALL:ALL) ALL
```

Configuring Sudo for Born2beroot

For security reasons, the Born2beroot subject requires a few more sudo configurations. We must:

- limit authentication for sudo to 3 tries for a bad password,
- define a custom message in case of a bad password,
- log sudo commands in `/var/log/sudo/`,
- and activate TTY to stop malicious software from giving itself root privileges via sudo.

In order to do this, we need to add the following lines to the `sudoers.tmp` file (as above, we must open it from the root session, with the **`sudo visudo`** command to be able to write changes to it):

```
Defaults     passwd_tries=3
Defaults     badpass_message="Wrong password. Try a
Defaults     logfile="/var/log/sudo/sudo.log"
Defaults     log_input
Defaults     log_output
Defaults     requiretty
```

If the `/var/log/sudo` directory doesn't exist, we might have to **`mkdir sudo`** in `/var/log/`.

Now, we can have root privileges in secure way, without having to log into the root session.

AppArmor pour Born2beroot Debian

AppArmor is a security program for Linux distributions that allows a system administrator to rescrict a program's access to the operating system. Each program has an associated profile that controls its ability to access the network, its read, write and execution permissions, among other things.

AppArmor is considered an alternative to **SELinux**, a similar program but much more difficult to install and maintain. The two programs work in opposite ways. AppArmor tends to allow everything and then gradually restrict, whereas SELinux starts off by restricting everything, and then loosening up little by little. But that's not the only difference: SELinux applies labels to each file, whereas AppArmor monitors software paths.

Despite its difficulties of use, SELinux is probably more secure than AppArmor. We could install SELinux on our Debian machine, but AppArmor came preinstalled and the Born2beroot subject explicitely states that "AppArmor for Debian must be running at startup".

To check that AppArmor is indeed installed on our system, let's ask to see its status:

```
$ sudo aa-status
```

If AppArmor is there and working correctly, we should see something like this:

```
apparmor module is loaded.
3 profiles are loaded.
...
```

UFW pour Born2beroot

UFW (*Uncomplicated **Firewall***) is, as its name suggests, a simple command-line firewall. A firewall is a program that monitors and controls data traffic between a local computer and the network at large. It decides whether to allow or block the traffic according to a set of security rules.

IP Addresses and Ports

When we talk about traffic and networks, we must understand two things: IP addresses and ports. On the internet, data is transferred from one computer to the other using their IP addresses, which look something like this: 109.234.160.5.

But to avoid conflicts between the various internet protocols, every computer separates the access paths thanks to specified ports, marked after the IP address, like this: 109.234.160.5:80 (port #80). Website data, transferred via HTTP uses port 80. HTTPS uses port 443, SSH port 22, SMTP (outgoing email) port 25, IMAP (incoming email) port 143, etc.

Configuring UFW for Born2beroot

With UFW, we will be able to define some rules governing access and restrictions, port by port. Let’s install and start UFW:

```
$ sudo apt update
$ sudo apt upgrade
$ sudo apt install ufw
$ sudo ufw enable
```

If everything goes well, we should get a message stating that the firewall is active and enabled at system startup. At any time, we can check the status of UFW and that of the ports with the following command:

```
$ sudo ufw status verbose
```

We can notice here that by default, the rule for outgoing traffic if authorize. If we forbid outgoing traffic by default with the following command, the packet manager APT and other essential programs will stop working correctly:

```
$ sudo ufw default deny outgoing
$ sudo apt update
$ sudo ufw default allow outgoing
```

The APT update shows error messages because it can no longer use its outgoing port in order to find the list of packages. So we must allow outgoing traffic by default.

For now, we have no rules for specific ports in our list when we check UFW’s status. We will soon! We can try authorizing or denying the port 4242 for example with the following commands:

```
$ sudo ufw allow 4242
$ sudo ufw deny 4242
```

Deleting a UFW Rule

To delete a rule, as we will be asked to do during the project evaluation, there are two options. The first is a simple command. Let’s say we have two rules in our list that we want to delete, “allow 4242” and “deny 4343”:

```
$ sudo ufw delete allow 4242
$ sudo ufw delete deny 4343
```

Those rules are now deleted.

The other way of doing it is to show the list of rules with an index. Then, we can delete a rule by its index number. Careful though, if we wish to delete several rules this way: the index numbers may change after a deletion.

```
$ sudo ufw status numbered
$ sudo ufw delete 2
```

And that’s it for UFW. Pretty uncomplicated, right?

SSH for Born2beroot

SSH, or **S**ecure **Sh**ell, is a network protocol that allows us to connect to a computer from a distance, over an unsecured network. Thanks to user passwords and the creation of public and private keys, the data communicated from one computer to the other is encrypted.

Configuring OpenSSH for Born2beroot

In Born2beroot, we are asked to install this protocol and route it trough the 4242 port. **OpenSSH** is the most popular and widespread tool, so let's install that one. Since we want to be able to connect to our Born2beroot machine from another machine, we need the **`openssh-server`** packet. In order to connect to another machine from the Born2beroot machine, we would need the `openssh-client` packet, but that isn't the case here.

```
$ sudo apt update
$ sudo apt upgrade
$ sudo apt install openssh-server
```

Now, let's run the following command to check the status of our SSH server. We should see "active" in green:

```
$ sudo systemctl status ssh
```

Here we can see that the server SSH is listening to port 22. We need to change that to port 4242. That can be done by editing the ssh configuration file:

```
$ sudo nano /etc/ssh/sshd_config
```

The line we are looking for is towards the beginning of the file and reads "#Port 22". We want to uncomment that and change it to "Port 4242" (no #!). Then, we have to restart the ssh service for the change to take effect.

```
$ sudo systemctl restart ssh
```

We can check the SSH server status again, and at the bottom we should see "Server listening on :: port 4242".

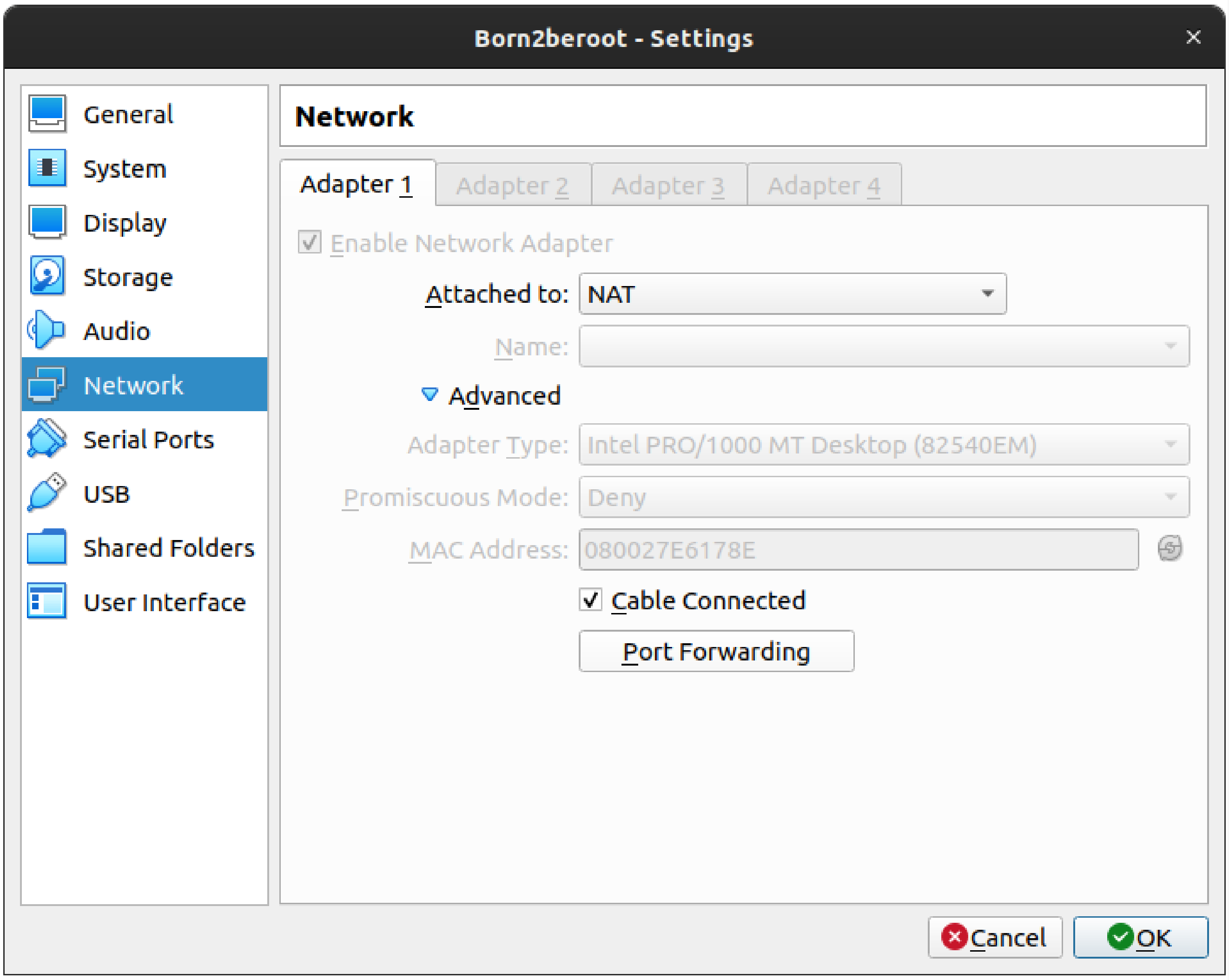
Let's not forget to tell our firewall to authorize the 4242 port connection! We might also have to delete a new rule about Port 22 which was added automatically with OpenSSH's installation.

```
$ sudo ufw allow 4242
$ sudo delete allow ssh #To delete port 22 rule
```

Port Forwarding in VirtualBox

Before we can connect to the virtual machine from another computer via SSH, we have to make a little adjustment in VirtualBox. Indeed, the connection will be refused until we forward the host port to the VM port.

In VirtualBox, select the Born2beroot machine and go into configuration settings.



☑ Enable Network Adapter
Attached to: NAT
Name:
Advanced
Adapter Type: Intel PRO/1000 MT Desktop (82540EM)
Promiscuous Mode: Deny
MAC Address: 080027E6178E
☑ Cable Connected

Then, go to **network** >> **Adapter 1** >> **Advanced** >> **Port Forwarding**. Then we will redirect the host port 4242 to the guest port 4242 like this:

Finally, in the virtual server Born2beroot, we are going to restart the SSH server once again and check its status:

```
$ sudo systemctl restart ssh
$ sudo systemctl status ssh
```

Logging into the Born2beroot Server via SSH

Now that we have configured everything, we can check the SHH connection by attempting to log into the Born2beroot virtual machine from the host machine terminal. Of course, the virtual machine must be turned on to be able to connect to it.

From the host machine's terminal, we can connect via SSH with this command:

```
$ ssh <username_server>@<server_IP_address> -p <ssl
```

The username will of course be that of the virtual machine user and the port will be 4242. But what is the IP address of our Born2beroot server? Since the virtual machine shares the host's IP address, we can simply use the localhost IP address. Localhost is an internal shortcut that every computer uses to refer to their own IP address. The localhost IP is 127.0.0.1.

So we can transcribe the previous command in one of the two following ways:

```
$ ssh mcombeau@localhost -p 4242
$ ssh mcombeau@127.0.0.1 -p 4242
```

Once we enter the user password, we can control the virtual machine from the outside! Let's note that the command prompt has changed and now shows the virtual machine's hostname.

In order to put an end to the SSH connection, all we need to do is:

```
$ exit
```

Password Policy for Born2beroot

Born2beroot asks us to set up a strong password policy. It must enforce the following rules:

1. Passwords have to expire every 30 days.
2. 2 must be the minimum number of days before being allowed to change a password.
3. The user must receive a warning message 7 days before their password expires.
4. A password must be at least 10 characters long.
5. It must contain an uppercase letter and a number. Also, it must not contain more than 3 consecutive identical characters.
6. The password must not include the name of the user.
7. The following rule does not apply to the root password: The password must have at least 7 characters that are not part of the former password.

In order to configure the first 3 rules, we have to edit the `/etc/login.defs` file:

```
$ sudo nano /etc/login.defs
```

And find the "*Password aging controls*" section to change the values such that:

```
PASS_MAX_DAYS 30
PASS_MIN_DAYS 2
PASS_WARN_AGE 7
```

However, these changes will not automatically apply to preexisting users. For both root and our first user, we need to use the **`chage`** command to enforce these rules. The **`-l`** flag is available to display which rules apply to a certain user.

```
$ sudo chage -M 30 <username/root>
$ sudo chage -m 2 <username/root>
$ sudo chage -W 7 <username/root>
$ sudo chage -l <username/root>
```

For the rest of the rules, we will have to install the password quality verification library.

```
$ sudo apt install libpam-pwquality
```

Then, we must edit the `/etc/security/pwquality.conf` configuration file. Here, we will need to remove the comment sign (#) and change the values of the various options. We will end up with something like this:

```
# Number of characters in the new password that mus
# old password.
difok = 7
# The minimum acceptable size for the new password
# credits are not disabled which is the default)
minlen = 10
# The maximum credit for having digits in the new p
# it is the minimun number of digits in the new pas
dcredit = -1
# The maximum credit for having uppercase characte
# If less than 0 it is the minimun number of upper
# password.
ucredit = -1
# ...
# The maximum number of allowed consecutive same ch
# The check is disabled if the value is 0.
maxrepeat = 3
# ...
# Whether to check it it contains the user name in
# The check is disabled if the value is 0.
usercheck = 1
# ...
# Prompt user at most N times before returning with
retry = 3
# Enforces pwquality checks on the root user passwo
# Enabled if the option is present.
```

```
enforce_for_root
# ...
```

And that’s all there is to it!

The last thing to do is to change the passwords of the preexisting user and that of root, to comply with the new password policy:

```
$ sudo passwd <user/root>
```

Hostname, Users and Groups

During Born2beroot’s defense, we will be asked to change our server’s hostname, to create a new user and handle user group modifications. For that, we will need to know the following commands.

Changing the Hostname

The hostname of our virtual machine must be our **intra login + 42**. Whether we correctly named it during the installation or not, we will have to change it during the evaluation. So we must know how. The following command will do the trick:

```
$ sudo hostnamectl set-hostname <new_hostname>
```

We could also change the hostname by editing the `/etc/hostname` file instead.

For the changes to take effect, we must restart the machine, which may take some time. The alternative is simply to show the hostname status after the change:

```
$ hostnamectl status
```

User Management

At startup, there must be at least two users: root and a personal user with our **intra login** as a username. For the evaluation, we must also be able to show a list of all users, add or delete user accounts, change their usernames, add or remove them from groups, etc. The following commands are necessary to do all of that:

- `useradd` : creates a new user.
- `usermod` : changes the user's parameters: `-l` for the username, `-c` for the full name, `-g` for groups by group ID.
- `userdel -r` : deletes a user and all associated files.
- `id -u` : displays user ID.
- `users` : shows a list of all currently logged in users.
- `cat /etc/passwd | cut -d ":" -f 1` : displays a list of all users on the machine.
- `cat /etc/passwd | awk -F '{print $1}'` : same as above.

Group Management

In the same way, we will have to manage user groups. Our default personal user must be in the **sudo** and **user42** groups. The following commands need to be mastered for the evaluation:

- `groupadd` : creates a new group.

- **`gpasswd -a`** : adds a user to a group.
- **`gpasswd -d`** : removes a user from a group.
- **`groupdel`** : deletes a group.
- **`groups`** : displays the groups of a user.
- **`id -g`** : shows a user's main group ID.
- **`getent group`** : displays a list of all users in a group.

Monitoring.sh for Born2beroot

The last thing we need to complete the mandatory part of Born2beroot is a small bash script. This script, in a file named `monitoring.sh`, must display the following information every 10 minutes on every terminal, from the time the server starts up:

- Operating system's architecture and its kernel version.
  - **`uname -srvmo`**
- Number of physical processors.
  - **`grep 'physical id' /proc/cpuinfo | uniq | wc -l`**
- Number of virtual processors.
  - **`grep processor /proc/cpuinfo | uniq | wc -l`**
- Current available RAM on the server and its utilization rate as a percentage
  - Used : **`free -h | grep Mem | awk '{print $3}'`**
  - Total : **`free -h | grep Mem | awk '{print $2}'`**
  - Percentage : **`free -k | grep Mem | awk '{printf("%.2f%%"), $3 / $2 * 100}'`**
- Current available memory on the server and its utilization rate as a percentage..
  - Used : **`df -h --total | grep total | awk '{print $3}'`**
  - Total : **`df -h --total | grep total | awk '{print $2}'`**

    - Percentage : `df -k --total | grep total | awk '{print $5}'`
- Current utilization rate of the processors as a percentage.
    - `top -bn1 | grep '^%Cpu' | cut -c 9- | xargs | awk '{printf("%.1f%%"), $1 + $3}'`
- Date and time of the last reboot.
    - `who -b | awk '{print($3 " " $4)}'`
- Whether LVM is active or not.
    - `if [ $(lsblk | grep lvm | wc -l) -eq 0 ]; then echo no; else echo yes; fi`
- Number of active connections.
    - `grep TCP /proc/net/sockstat | awk '{print $3}'`
- Number of users using the server.
    - `who | wc -l`
- Server's IPv4 address and its MAC (Media Access Control) address.
    - IP : `hostname -I | awk '{print $1}'`
    - MAC : `ip link show | grep link/ether | awk '{print $2}'`
- Number of commands executed with the sudo program.
    - `grep COMMAND /var/log/sudo/sudo.log | wc -l`

Some of the above commands won't work without root permissions. Which is why we will log in as root and create the `monitoring.sh` file there. We must also grant this file execution rights with:

```
# chmod 755 monitoring.sh
```

In order to be able to execute the script as required, we must understand how to broadcast a message to every terminal and how to automate the script so that it executes every 10 minutes.

The Wall Command

The **wall** command is the one that allows us to broadcast a message to all users at a time, in all terminals. It can receive either some text or the contents of a file. By default, the announcement is prefixed with a banner, but that banner is optional in this project.

There are two options:

- **`wall "message"`**
- **`wall -n "message"`**: displays without the banner

The Cron Service

Cron (or **cron**tab, short for **chrono** **tab**le) is a program that enables the execution of scripts or software in an automatic way, at a certain date and time or at a specified interval. It is installed by default in Debian (we can check this with the `apt list cron` command). To be certain it will run at system startup, we should enable it:

```
# systemctl enable cron
```

Cron uses `crontab` files to schedule jobs. Each user can have one, or many. As the root user, we will now create one with the following command:

```
# crontab -e
```

The syntax of a cron file might seem obscure, but it's not too hard to wrap your head around:

```
* * * * * <command to execute>
```

Here, the stars represent temporal values:

```
.------------------------ minutes (0-59)
| .-------------------- hours (0-23)
| | .---------------- day of the month (1-31)
| | | .------------ month of the year (1-12)
| | | | .-------- day of the week (0-6, 0 = sunday
| | | | |
* * * * * <command to execute>
```

By replacing the stars with numerical values, we can define when our command must be executed.

So is this the way we should write “every 10 minutes”?

```
10 * * * * bash /root/monitoring.sh
```

Almost, but no. This instruction means “execute this at the tenth minute of each hour, every day of every month”. So our monitoring script won’t execute every 10 minutes, but only at midnight 10, 1:10, 2:10, 3:10, 4:10 and so on.

So how are we supposed to say “every 10 minutes”, then? Well, we can simply “divide” the minute star by 10.

```
*/10 * * * * bash /root/monitoring.sh
```

If the wall command wasn’t incorporated directly into the `monitoring.sh` script, we can pipe it into this cron rule, like so:

```
*/10 * * * * bash /root/monitoring.sh | wall
```

However, the script is now being executed every ten minutes of the hour, not every ten minutes **from boot**, as the subject requires. To create the proper delay, we can use the **sleep** command which pauses a command for a certain number of seconds.

Creating a Sleep Delay

Let's create a new script called **sleep.sh**. In this script, we will calculate the number of seconds between the precise boot time and the tenth minute. For this to work, we will use bc, the basic command-line calculator, which we can install with **apt install bc**. Then, in `sleep.sh`:

```
#!bin/bash

# Get boot time minutes and seconds
BOOT_MIN=$(uptime -s | cut -d ":" -f 2)
BOOT_SEC=$(uptime -s | cut -d ":" -f 3)

# Calculate number of seconds between the nearest 
# Ex: if boot time was 11:43:36
# 43 % 10 = 3 minutes since 40th minute of the hou
# 3 * 60 = 180 seconds since 40th minute of the ho
# 180 + 36 = 216 seconds between nearest 10th minu
DELAY=$(bc <<< $BOOT_MIN%10*60+$BOOT_SEC)

# Wait that number of seconds
sleep $DELAY
```

We can now modify our crontab to take the `sleep.sh` delay into account before executing `monitoring.sh` :

```
*/10 * * * * bash /root/sleep.sh && bash /root/mon
```

There, our `monitoring.sh` script will now be executed every 10 minutes **from the machine startup time**.

Some Final Tips

ERROR Failed to send host log message

At machine boot, we may notice the following error: [DRM :vmw_host_log [VMWGFX]] *ERROR* Failed to send host log message. It is a small issue with the graphics controller which does not affect the machine's operation. However, it isn't very nice to see. We can easily solve this error by changing our graphics controller:

- Turn off the virtual machine,
- Go to VirtualBox >> Machine >> **Settings**,
- Go to **Display** >> **Screen** >> **Graphics Controller**,
- Choose **VBoxVGA**.

There should no longer be an error when we reboot the machine!

Signature.txt

The Born2beroot subject explains how to extract the virtual machine's signature. However, we must realize that this signature will change the minute we make any modification to the virtual machine. This means that the signature will be correct for the first evaluation, but will have changed before the second. It is therefore very important to take a snapshot of the machine right before extracting the signature, and to restore the machine from that snapshot between evaluations.

And that's all for the Born2beroot mandatory part. All that's left is to do the bonuses!

Bonuses for Born2beroot

The Born2beroot bonuses invite us to install a WordPress website as well as another service of our own choice. However, this will be covered in a third and last article, this one is way too long already!

Born2beroot : Installation | **Configuration** | Bonus | Subject [pdf]

Sources and Further Reading

- Gunjit Khera, *What is APT and Aptitude? and What's real Difference Between Them?* [tecmint]
- Debian Wiki, *sudo* [Debian Wiki]
- Linuxize, *How to Add User to Sudoers in Debian* [Linuxize]
- Justin Ellingwood, Brian Boucheron, *How to Edit the Sudoers File* [DigitalOcean]
- Wikipédia, *AppArmor* [Wikipédia]
- Tuyen Pham Thanh, *AppArmor vs SELinux* [Omarine]
- TechTerms, *Port* [TechTerms]
- Port Forward, *What is a Port?* [Port Forward]
- Debian Wiki, *Uncomplicated Firewall (ufw)* [Debian Wiki]
- Debian Wiki, *SSH* [Debian Wiki]
- Bradley Mitchell, *127.0.0.1 IP Address Explained* [LifeWire]
- Daniel López Azaña, *Differences between physical CPU vs logical CPU vs Core vs Thread vs Socket* [Daniloaz.com]
- Manuel du programmeur Linux :
  - *uname (1)* [man]
  - *free (1)* [man]

  - *df (1)* [man]
  - *who* (1) [man]
- Christopher Murray, *Understanding Crontab in Linux With Examples* [Linux Handbook]
- Wikipédia, *cron* [Wikipédia]
- James Timmerwilke, *What is a vCPU and How Do You Calculate vCPU to CPU?* [Datacenters.com]

42 cursus APT computer crontab Debian SSH sudo UFW virtual machine

ABOUT THE AUTHOR

Mia Combeau

Student at 42Paris, digital world explorer. I code to the 42 school norm, which means for loops, switches, ternary operators and all kinds of other things are out of reach… for now!

VIEW ALL POSTS

ADD COMMENT

Comment

Name *

Email *

Website

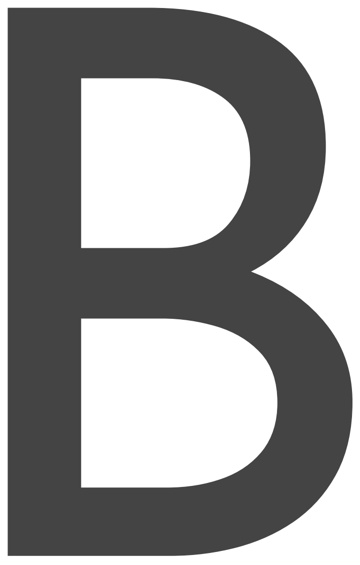
☐ Save my name, email, and website in this browser for the next time I comment.

SUBMIT COMMENT

READ MORE

Binary 010: The Uses of Bit Shifting and Bitwise Operations

May 7, 2022



Pipex: Reproducing the Pipe Operator “|” in C

April 2, 2022

P

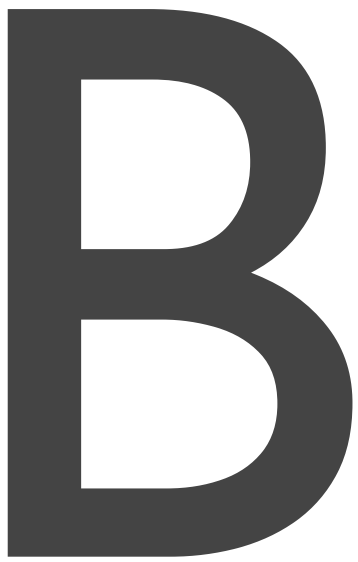
Born2beroot 03: Installing WordPress on a Debian Server

March 13, 2022

B

Born2beroot 01: Creating a Debian Virtual Machine

March 9, 2022



By Mia Combeau March 11, 2022

To Top

Born2beroot 03: Installing WordPress on a Debian Server

To Top

Born2beroot 01: Creating a Debian Virtual Machine

MENU

- Legal Notice
- Contact

CATEGORIES

- 42 School Projects
- C Programming
- Computer Science

codequoi

- HOME
- CATEGORIES

Type here to search... SEARCH

CATEGORIES

- 42 School Projects
- C Programming
- Computer Science

MIA COMBEAU

Student at 42Paris, digital world explorer. I code to the 42 school norm, which means for loops, switches, ternary operators and all kinds of other things are out of reach... for now!

- github
- linkedin