Debian

Setup Crontab on Debian 12

3 months ago • by Sidratul Muntaha

In this guide, we will demonstrate how to setup crontab on Debian 12.

Prerequisites:

To perform the steps that are demonstrated in this guide, you need the following components:

- A properly-configured Debian system. Learn more about installing Debian.
- If you're currently running an older release, check out how to upgrade to Debian 12.

Crontab on Debian

In UNIX/Linux, cron is a command-line utility that can run the scheduled jobs at a specified time, date, or interval. The cron daemon starts at boot and handles the execution of the scheduled jobs. It's a simple scheduling tool that most UNIX/Linux systems come with preinstalled (including Debian).

There are a couple of cron-related concepts that you should know about:

- **crontab**: It's an abbreviation of the term "cron table". It's a system file that is structured like a table. Within the file, all the scheduled jobs are described (with specific time or interval).
- **crond**: It's the cron daemon that runs in the background. The daemon starts at system startup and runs the various tasks that are described in the crontab.
- **cron** jobs: In the context of cron, each scheduled tasks are referred to as "jobs".

Note that cron uses **/bin/sh** as the default shell.

Crontab File Locations

There are multiple crontab files available throughout the system:

- **/etc/crontab**: The main system crontab
- **/var/spool/cron/**: It's a directory that contains all the user-specific crontab
- **/etc/cron.d/**: It's a directory that contains all the system crontab

Cron Permissions

Any cron job runs under a specific user. Thus, each job inherits the user permission of the owner.

For example, a normal user test isn't permitted to run the jobs that require a root permission. However, the root user can issue the jobs that can perform anything on the system. For example, updating the packages periodically.

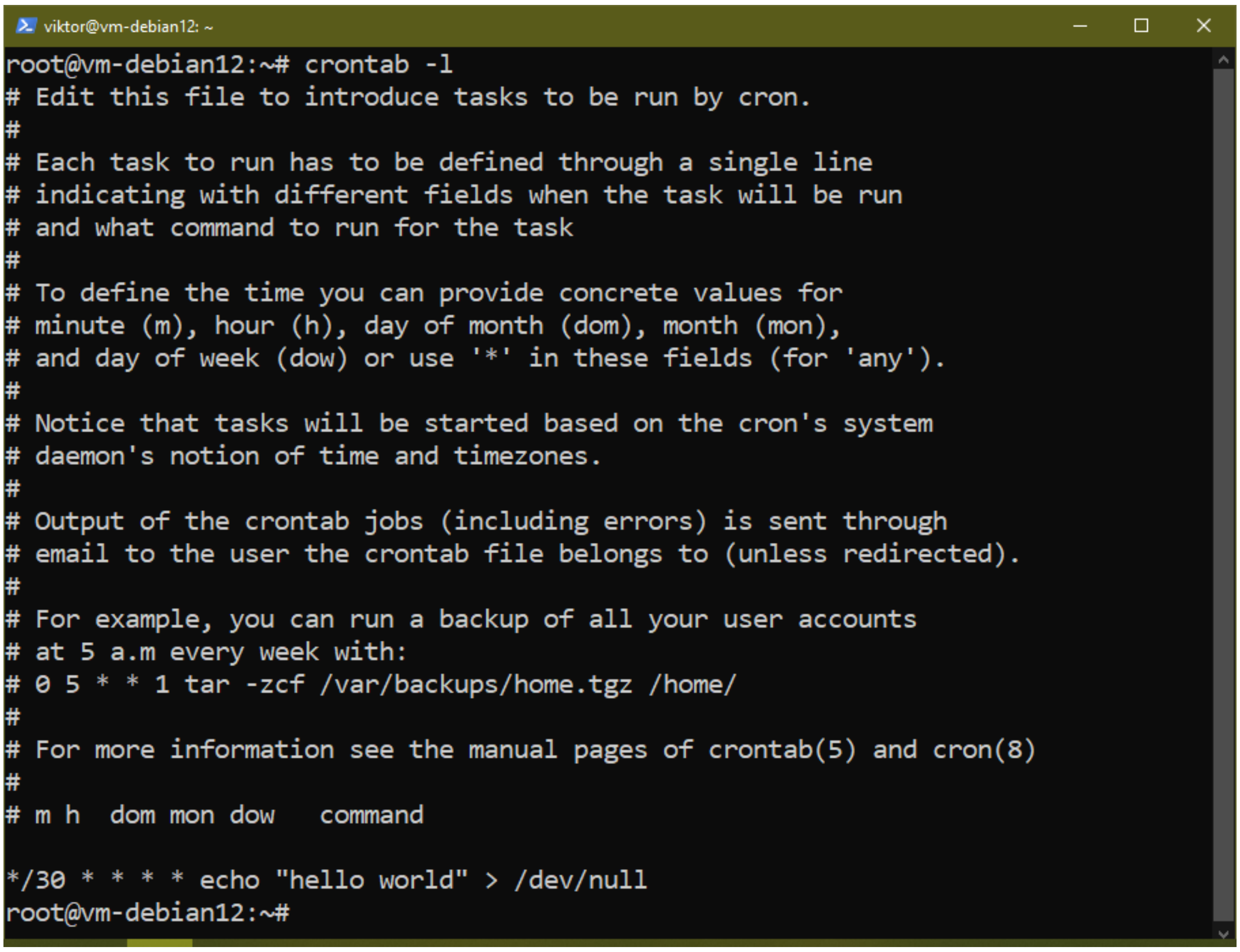
Configuring Crontab

In this section, we will learn about working with crontab.

Viewing the Crontab

While we can directly manipulate the crontab files from the location that is specified before, it's strongly recommended to use the "crontab" command to ensure stability and compatibility. To check the content of the crontab, run the following command:

```
$ crontab -l
```



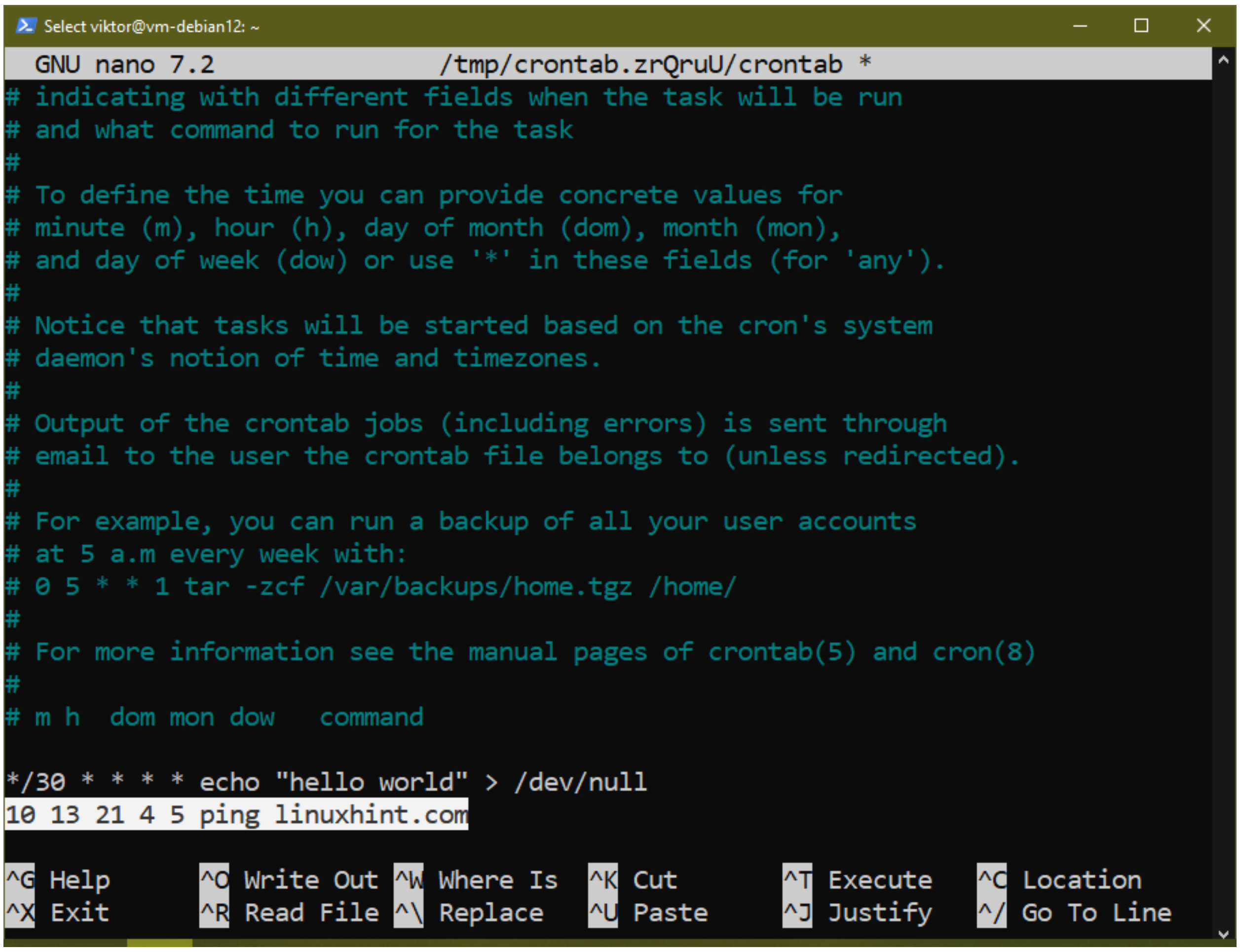
```
root@vm-debian12:~# crontab -l
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# 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

*/30 * * * * echo "hello world" > /dev/null
root@vm-debian12:~#
```

It prints out the entire crontab file of the specific user.

Crontab Syntax

The crontab syntax is better described using an example:

```
$ 10 13 21 4 5 ping linuxhint.com
```



```
  GNU nano 7.2                /tmp/crontab.zrQruU/crontab *
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# 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

*/30 * * * * echo "hello world" > /dev/null
10 13 21 4 5 ping linuxhint.com

^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line
```

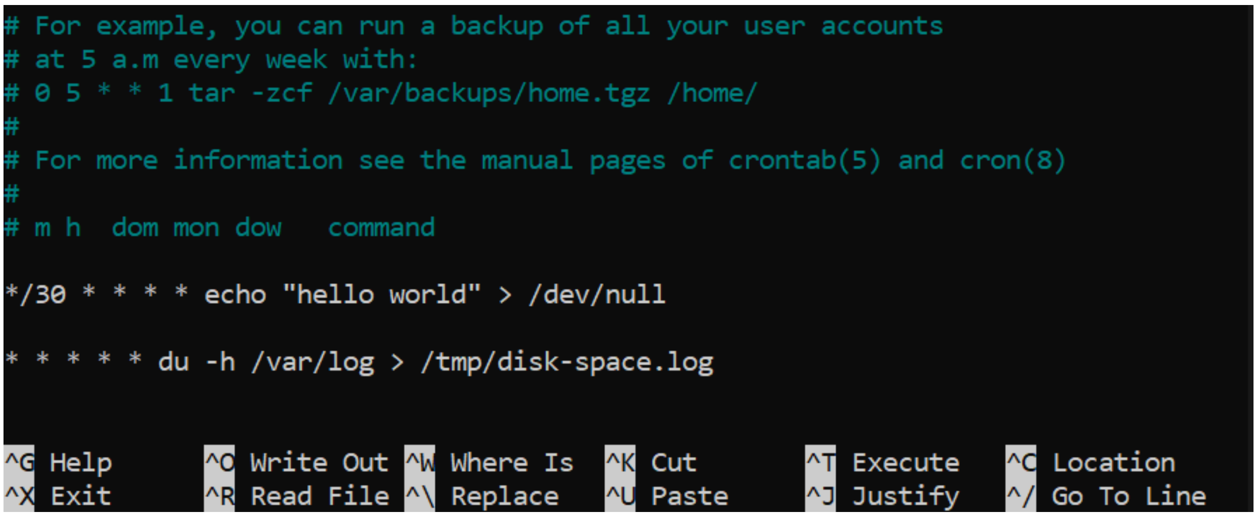
Here:

- **10**: It’s the minute field. The value can be 0-59 or asterisk (*) which denotes every minute.
- **13**: It’s the hour field. The value can be 0-23 or asterisk (*) which denotes every hour.
- **21**: It denotes the day of the month. The value can be 0-31 or asterisk (*) which denotes every month.
- **4**: It denotes the month of the year. The value can be 1-12 or asterisk (*) which denotes every year.
- **5**: It denotes the day of the week. The value can be 0-6 or asterisk (*) which denotes every day of the week. Note that the week starts with Sunday.
- **ping linuxhint.com**: At the specified time, cron runs the described command.

In short, cron pings the linuxhint.com host on Friday, 21st March at 13:10.

Let’s put this knowledge in action. In the next example, we will monitor the disk space usage of **/var/log** every minute and write the result in a log:

```
$ * * * * * du -h /var/log > /tmp/disk-space.log
```

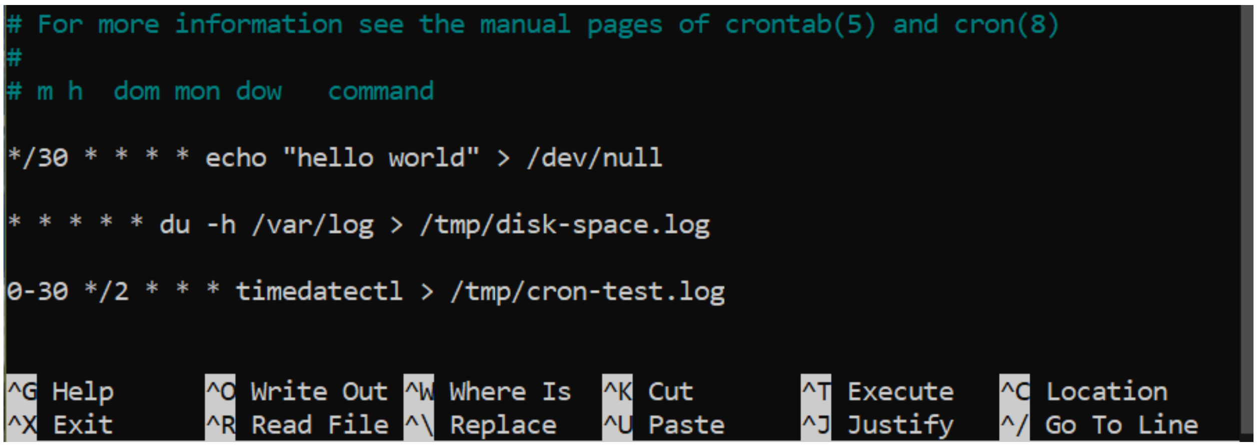


```
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command

*/30 * * * * echo "hello world" > /dev/null

* * * * * du -h /var/log > /tmp/disk-space.log
```

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line

Cron also supports the ranged and stepped values. Check out the following examples:

```
$ 0-30 */2 * * * <command_or_script>
```



```
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command

*/30 * * * * echo "hello world" > /dev/null

* * * * * du -h /var/log > /tmp/disk-space.log

0-30 */2 * * * timedatectl > /tmp/cron-test.log
```

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line

Here, the cron job runs each minute, for 30 minutes, every 2 hours.

There are also some special time syntaxes:

- **@reboot**: The job is run after each system boot.
- **@hourly**: The job runs at the beginning of each hour.
- **@daily**: The job runs every day at 00:00.
- **@weekly**: The job runs each week at Sunday.
- **@monthly**: The job runs at the beginning of each month.
- **@yearly**: The job runs at the beginning of each year.

Having trouble with writing your own cron syntax or need help debugging? There are some interactive tools like crontab.guru that dramatically simplifies the process.

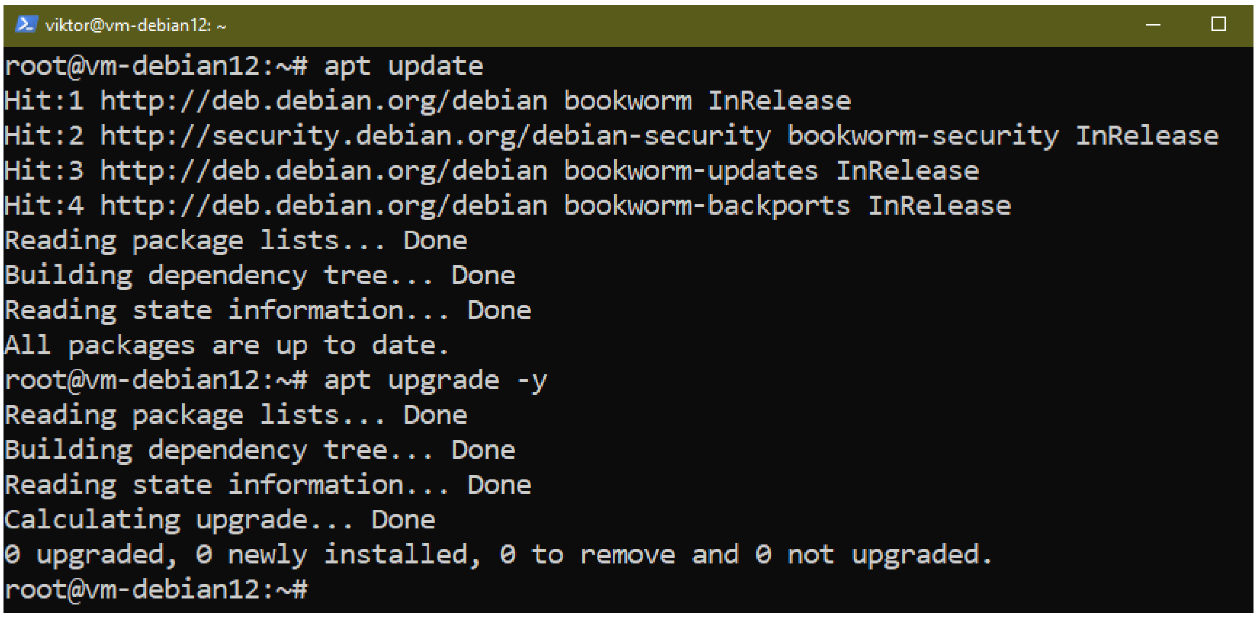
Crontab Example

This section features a handful of cron job examples.

Example 1: Auto Update the System

In Debian, to update all the installed packages, run the following commands:

```
$ sudo apt update
$ sudo apt upgrade -y
```



```
root@vm-debian12:~# apt update
Hit:1 http://deb.debian.org/debian bookworm InRelease
Hit:2 http://security.debian.org/debian-security bookworm-security InRelease
Hit:3 http://deb.debian.org/debian bookworm-updates InRelease
Hit:4 http://deb.debian.org/debian bookworm-backports InRelease
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
All packages are up to date.
root@vm-debian12:~# apt upgrade -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
root@vm-debian12:~#
```

We can use crontab to automate this process. Making system changes require root permission, so we put the job under root.

Change the current user to root:

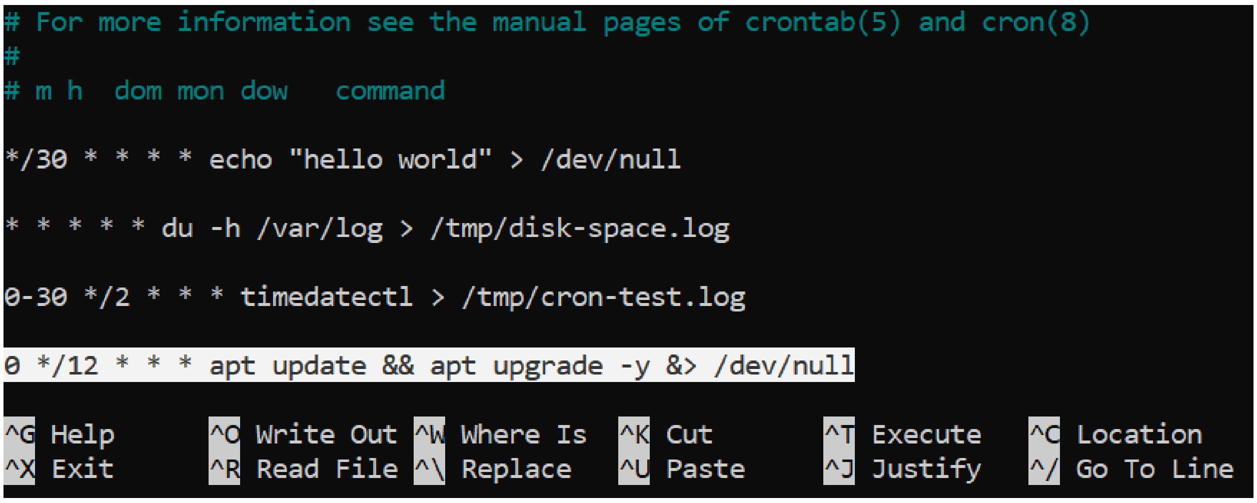
```
$ su -
```

Now, launch the crontab editor:

```
$ crontab -e
```

The following cron job automatically checks for updates twice a day:

```
$ 0 */12 * * * apt update && apt upgrade -y &> /dev/null
```



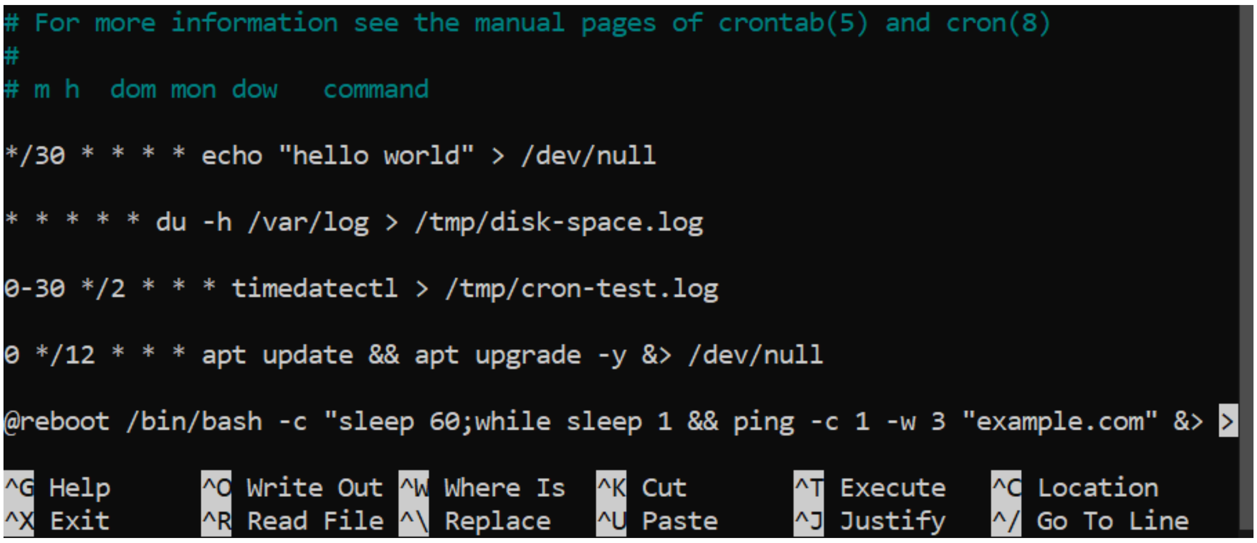
```
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command

*/30 * * * * echo "hello world" > /dev/null

* * * * * du -h /var/log > /tmp/disk-space.log

0-30 */2 * * * timedatectl > /tmp/cron-test.log

0 */12 * * * apt update && apt upgrade -y &> /dev/null

^G Help       ^O Write Out ^W Where Is  ^K Cut        ^T Execute    ^C Location
^X Exit       ^R Read File ^\ Replace   ^U Paste      ^J Justify    ^/ Go To Line
```

Example 2: Auto Shutdown

We can use cron to auto shutdown the system when certain conditions are met. For example, a certain host is unavailable due to power outage.

Take a look at the following Bash script:

```
while sleep 1 && ping -c 1 -w 3 "example.com" &> /dev/null
do
    continue
done
/sbin/shutdown now
```



```
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command

*/30 * * * * echo "hello world" > /dev/null

* * * * * du -h /var/log > /tmp/disk-space.log

0-30 */2 * * * timedatectl > /tmp/cron-test.log

0 */12 * * * apt update && apt upgrade -y &> /dev/null

@reboot /bin/bash -c "sleep 60;while sleep 1 && ping -c 1 -w 3 "example.com" &> >

^G Help       ^O Write Out ^W Where Is  ^K Cut        ^T Execute    ^C Location
^X Exit       ^R Read File ^\ Replace   ^U Paste      ^J Justify    ^/ Go To Line
```

Here:

- We run an infinite "while"
- The "sleep" command controls the rate of execution of the loop (every 1 second).
- The "ping" command pings the host com.

- If the host is available, the loop continues. Since there's nothing else to do, it starts the next iteration.
- If the host is unavailable, the loop ends and subsequently executes the "shutdown"

We can transform the code into a single line:

```
$ while sleep 1 && ping -c 1 -w 3 "example.com" &> /dev/null; do continue; done;
/sbin/shutdown now
```

We can finally put the script into crontab:

```
$ @reboot /bin/bash -c "sleep 60;while sleep 1 && ping -c 1 -w 3 "example.com" &>
/dev/null;do continue;done;/sbin/shutdown now"
```

Here:

- We want the script to start running after the system boots.
- The additional "sleep" command at the beginning ensures that the system boots up properly before executing the script. Change the value as needed.
- Cron uses **/bin/sh** as the default shell. Since it's a Bash script, we invoke the Bash shell to run the script.

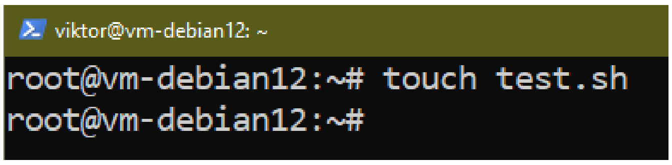
Example 3: Automated Execution of Scripts

From the previous example, it's clear that crontab entries can become extremely long, especially when it involves shell scripts. In addition, pruning scripts into a single line can be challenging, especially for big ones.

We can solve this issue by automating the launch of a shell script. With proper implementation, this technique can also dramatically reduce the number of required crontab entries.

To demonstrate, create a new shell script first:

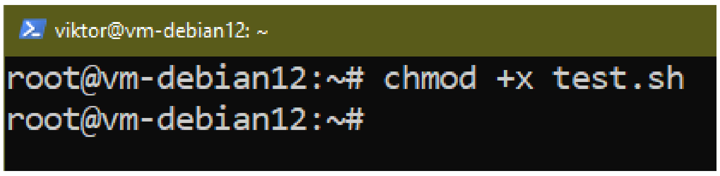
```
$ touch test.sh
```



```
root@vm-debian12:~# touch test.sh
root@vm-debian12:~#
```

Mark the file as an executable:

```
$ chmod +x test.sh
```

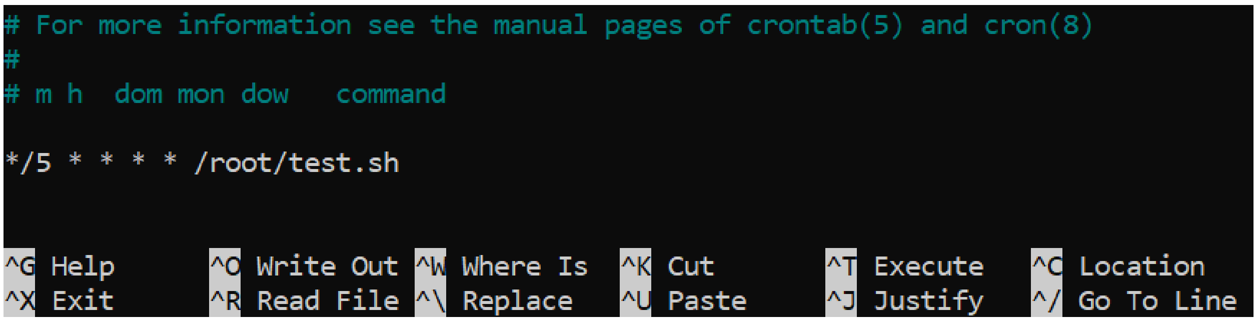

viktor@vm-debian12: ~

```
root@vm-debian12:~# chmod +x test.sh
root@vm-debian12:~#
```

You can place any shell script within the file. However, make sure to declare the proper shebang as it dictates what interpreter actually runs the code. Learn more about shebang Bash.

Finally, automate the execution of the script in crontab:

```
$ crontab -e
$ */5 * * * * <path_to_script>
```



```
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command

*/5 * * * * /root/test.sh


^G Help       ^O Write Out ^W Where Is  ^K Cut        ^T Execute    ^C Location
^X Exit       ^R Read File ^\ Replace   ^U Paste      ^J Justify    ^/ Go To Line
```

Conclusion

We demonstrated how to setup crontab on Debian 12. We discussed about various types of crontab files and their impacts. We also learned about the crontab automation syntax. Finally, we demonstrated how to automate various tasks using crontab.

For automation, shell scripting is another powerful tool. In Linux, Bash is the most popular shell. Check out Bash scripting for beginners. The Bash programming section also contains numerous additional guides on various aspects of Bash scripting.

Happy computing!