#!/bin/bash

touch /lib/systemd/system/startup.service

echo "" >> /lib/systemd/system/startup.service

cat >> /lib/systemd/system/startup.service <<EOL

[Unit]

Description=Startup Script

[Service]

ExecStart=/root/.local/bin/startup.sh

[Install]

WantedBy=multi-user.target

EOL

# vi /lib/systemd/system/startup.service

mkdir -p /root/.local/bin/

touch /root/.local/bin/startup.sh

chmod 755 /root/.local/bin/startup.sh

echo "" >> /root/.local/bin/startup.sh

cat >> /root/.local/bin/startup.sh <<EOF

#!/bin/bash

route add -net 192.168.20.0 netmask 255.255.255.0 gw 192.168.10.1

EOF

systemctl enable startup.service

**How to Run Script on Boot Up in Debian 11**

11 months ago

by [David Adams](https://linuxhint.com/author/david-adams/)

This tutorial explains how to run scripts or services at startup or boot in Debian 11 and based Linux distributions.

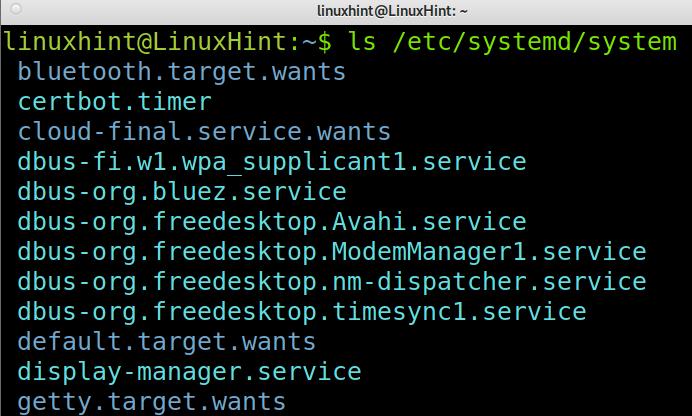
To properly add services and scripts at start up on Debian 11, you need to create Systemd units. This tutorial focuses on Systemd units, if you are looking for fast instructions to run a script at boot, jump straight to the Creating a Systemd unit to run a script at boot section.

**What is a Systemd unit?**

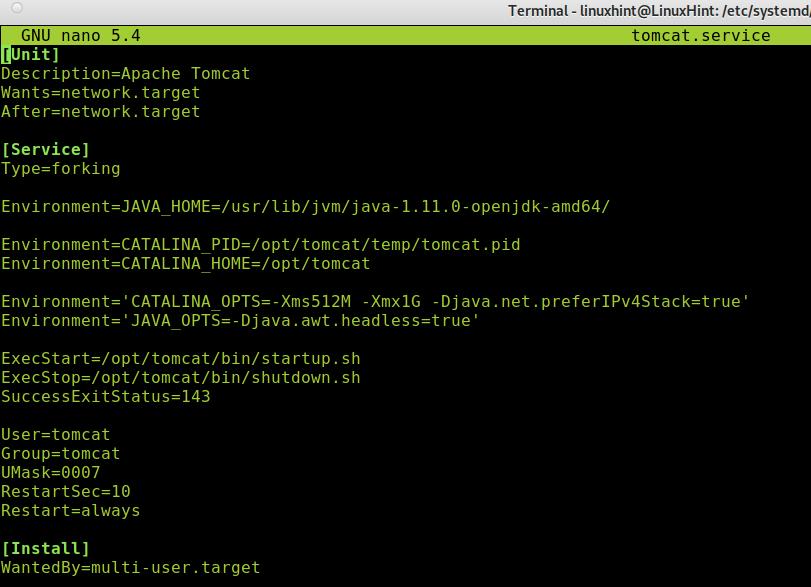
Systemd units are configuration files containing information for the proper management or execution of certain system files. Systemd Units can be used to manage services, sockets, devices, mount points, automount points, swap files or partitions, a start-up target, a watched file system path, timers controlled and supervised by systemd, a resource management slice or a group of externally created processes.

Systemd units are located under the directory */etc/systemd/system*. Any script you want to run at boot using Systemd units must be specified in a file (unit) under this directory.  
A fast view using the ls command will show us the existing systemd units in our system.

ls /etc/systemd/system



One of the files which were not included in the screenshot above is *tomcat.service* whose content seems like the following image.



**Note**: The following information is for you to get familiarized with unit configuration files. Don’t worry, the unit you’ll create to run your script will be simpler.

Where the [Unit] directives:

**Description**: This directive allows you to add a description for a unit, here you can set the unit name for it to be identified.

**Wants**: Here, you can specify the unit dependencies. Note there are different directives for this similar purpose. For example, the directive *Requires* is used to specify strict dependencies, without which the unit can’t work. Contrary to Requires, Wants is used to specify dependencies without which the unit can keep working.

**After**: The current unit will start after the unit specified in this directive.

[Service] section directives:

**Type**: In the previous example, forking indicates the service shall be killed while keeping child processes that must be assigned a PID.

**Environment**: Here, you can specify the unit environment variables.

**ExecStart**: This directive allows you to specify the path and commands you want to execute.

**ExecStop**: you can specify the command used to stop the unit.

**SuccessExitStatus**: This directive allows you to specify exit status and signals.

**User**: you can specify the user owner of the unit.

**Group**: you can specify the group owner of the unit.

**UMask**: you can specify the user mask.

**RestartSec**: If the unit restarts automatically, here you can specify the time to wait to retry restarting the service.

**Restart**: you can define for Systemd when the unit should be restarted. The available options are always, on-failure, on-abort, on-success, on-watchdog, and on-abnormal.

The [Install] directive in the example above is WantedBy.

**WantedBy**: This directive allows you to define the unit as dependency; it is similar to the *Wants* directive, but to specify if the current unit is considered a dependency by another unit.

Other common directives in the [Unit] section which weren’t included in the previous screenshot:

**Requires**: In this directive you can specify dependencies to prevent boot failures. Contrary to the *Wants* directive, if a dependency specified with the directive Requires isn’t met, the unit won’t work.

In the [Service] section:

**PIDFile**: The forking directive requires the PIDFile directive, which contains the path to the file pid of the child process for Systemd to identify it.

**StartLimitInterval**: indicates the unit has 60 seconds for 10 attempts to restart upon failure.

**StartLimitBurst**: This directive indicates the attempts limit, in the previous example, 10 attempts in 60 seconds.

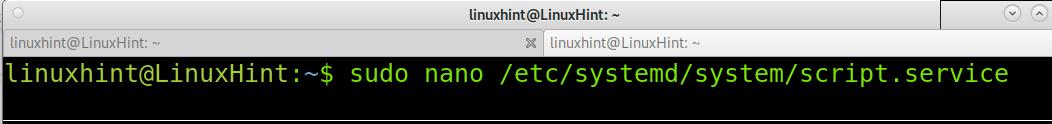
For additional information on Systemd units you can read:  
<https://manpages.debian.org/jessie/systemd/systemd.unit.5.en.html>.

**Creating a Systemd Unit to Run a Script at Boot on Debian 11**

Running a script at boot may be simpler and contain less configuration than the *tomcat.service* file previously used to show the unit’s common content.

As said previously, the files (units) containing the information on scripts running at boot are located in the directory */etc/systemd/system*. To define a script to run at boot, you need to create a new unit for this script. To create a unit under */etc/systemd/system*, you can use nano as shown in the example below, in which I create a unit named *script.service*, you can name it as you consider convenient to identify your script.

sudo nano /etc/systemd/system/script.service

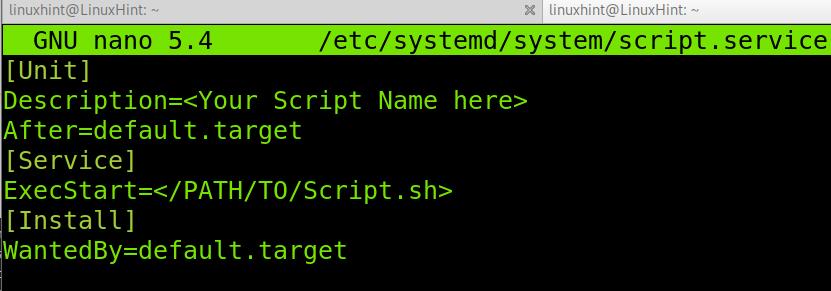


The content of the unit for your script will be simpler than the *tomcat.service* used as example previously.

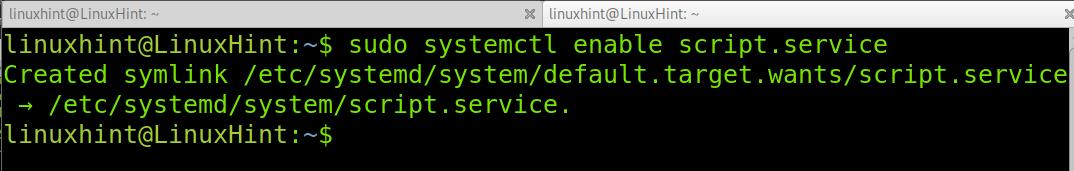
Copy and paste the following code into the file you created under */etc/systemd/system*.

**Note**: Replace *<Your Script Name here>* with your script name and *</PATH/TO/Script.sh>* with the path to your script.

[Unit]  
  
Description=Your Script Name here  
  
After=default.target  
  
[Service]  
  
ExecStart=/PATH/TO/Script.sh  
  
[Install]  
  
WantedBy=default.target



After copying the content into the file under */etc/systemd/system/*, you need to enable it using the systemctl command as shown below. Replace *<script.service>* with the correct name you assigned to the unit.



Now your script is ready to run at start up in Debian 11.

**Conclusion**

As you can see, running scripts at boot in Debian 11 using Systemd is pretty simple. Any Linux user level can get this task done by just creating a simple file within the units directory. Of course, Linux offers additional methods to run scripts at boot which were not described in this tutorial such as Cron. Any Linux user should know how to locate Systemd units to run services or required scripts at startup. Both Systemd units and cron are accepted methods, while rc.local or init.d scripts are obsolete.

Thank you for reading this tutorial explaining how to run scripts on boot using Systemd in Debian 11. Keep following us for additional Linux tips and tutorials.