

# Linux Fundamentals Part 3

---

Learned:

- The `ps` command shows running processes on a system, which can help identify programs that may contain malware, especially if the resource usage is unusually high.
- Monitoring access logs is important way to see any possible intrusion or many attempts to access a account by failed logins.
- Access logs can also show firewall activity, which helps identify if any are offline or any issues occurring.

## Terminal Text Editor's

### Nano

To create or edit a file using nano, we simply use `nano filename` -- replacing "filename" with the name of the file you wish to edit.

### Introducing Nano

```
tryhackme@linux3:/tmp# nano myfile
GNU nano 4.8 myfile

^G Get Help    ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur
Pos          M-U Undo      M-A Mark Text
^X Exit        ^R Read File  ^\ Replace    ^U Paste Text ^T To Spell   ^_ Go To
Line  M-E Redo      M-6 Copy Text
```

Once we press enter to execute the command, `nano` will launch Where we can just begin to start entering or changing our text. You can navigate each line using the "up" and "down" arrow keys or start a new line using the "Enter" key on your keyboard.

### Using Nano to write text

```
tryhackme@linux3:/tmp# nano myfile
GNU nano 4.8 myfile
Modified


Hello TryHackMe
I can write things into "myfile"

^G Get Help    ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur
```

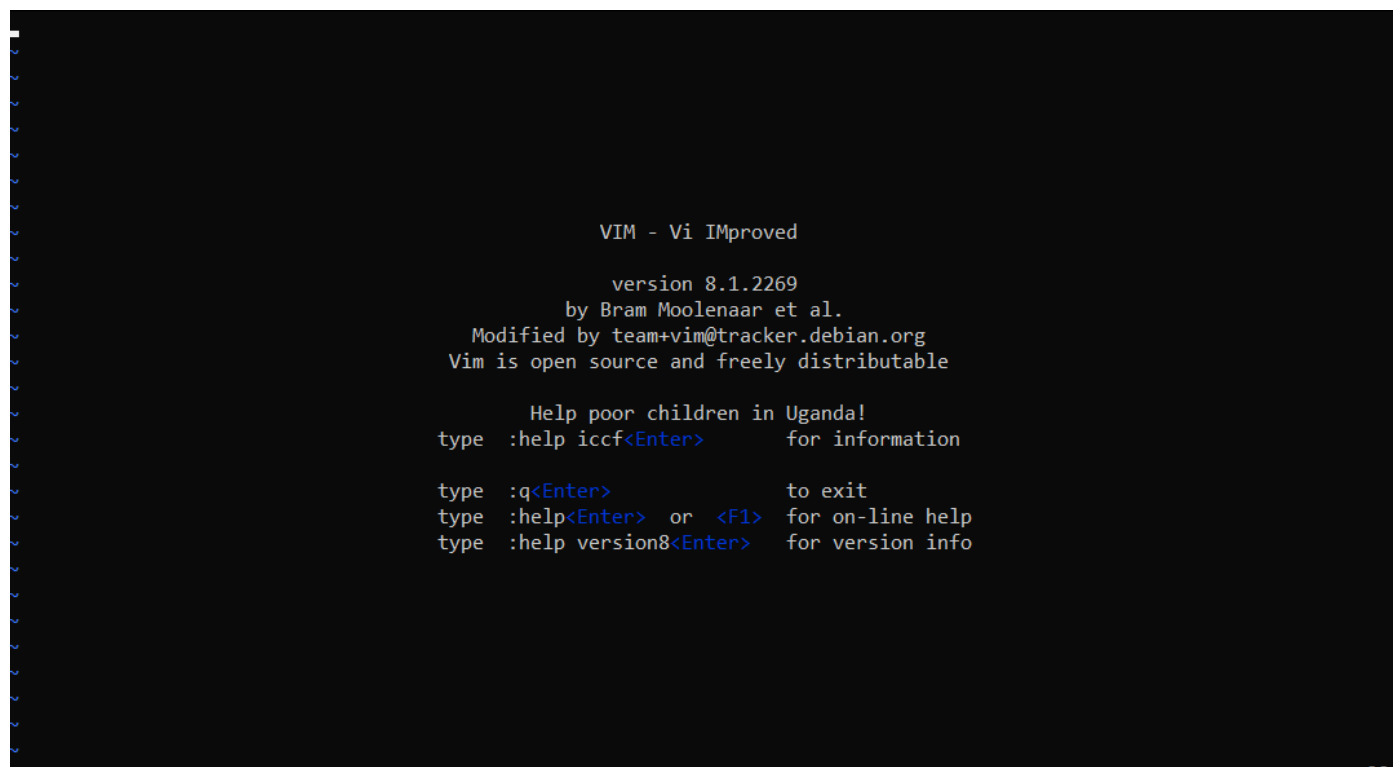
Pos	M-U Undo	M-A Mark Text			
<b>^X</b> Exit	<b>^R</b> Read File	<b>^\</b> Replace	<b>^U</b> Paste Text	<b>^T</b> To Spell	<b>^_</b> Go To
Line	M-E Redo	M-6 Copy Text			

Nano has a few features that are easy to remember & covers the most general things you would want out of a text editor, including:

- Searching for text
- Copying and Pasting
- Jumping to a line number
- Finding out what line number you are on

You can use these features of nano by pressing the **"Ctrl"** key (which is represented as an  on Linux) and a corresponding letter. For example, to exit, we would want to press **"Ctrl"** and **"X"** to exit Nano.

## VIM



```

VIM - Vi IMproved

version 8.1.2269
by Bram Moolenaar et al.
Modified by team+vim@tracker.debian.org
Vim is open source and freely distributable

Help poor children in Uganda!
type  :help iccf<Enter>      for information

type  :q<Enter>              to exit
type  :help<Enter> or <F1>   for on-line help
type  :help version8<Enter> for version info

```

Some of VIM's benefits are:

- Customizable - you can modify the keyboard shortcuts to be of your choosing
- Syntax Highlighting - this is useful if you are writing or maintaining code, making it a popular choice for software developers
- VIM works on all terminals where nano may not be installed
- There are a lot of resources such as [cheatsheets](#), tutorials, and the sorts available to you use.

# Downloading Files (Wget)

This command allows us to download files from the web via HTTP as if you were accessing the file in your browser. We simply need to provide the address of the resource that we wish to download. For example, if I wanted to download a file named "myfile.txt" onto my machine, assuming I knew the web address.

```
wget https://assets.tryhackme.com/additional/linux-fundamentals/part3/myfile.txt
```

## Transferring Files From Your Host - SCP (SSH)

Secure copy, or SCP, is a means of securely copying files. Unlike the regular cp command, this command allows you to transfer files between two computers using the SSH protocol to provide both authentication and encryption.

### Working on a model of SOURCE and DESTINATION, SCP allows you to

- Copy files & directories from your current system to a remote system
- Copy files & directories from a remote system to your current system

If we know the usernames and passwords for a user on my system and a user on the remote system.

Variable	Value
The IP address of the remote system	192.168.1.30
User on the remote system	ubuntu
Name of the file on the local system	important.txt
Name that we wish to store the file as on the remote system	transferred.txt

With this information, let's craft our `scp` command (remembering that the format of SCP is just SOURCE and DESTINATION)

```
scp important.txt ubuntu@192.168.1.30:/home/ubuntu/transferred.txt
```

And now let's reverse this and layout the syntax for using `scp` to copy a file from a remote computer that we're not logged into

Variable	Value
IP address of the remote system	192.168.1.30

User on the remote system	ubuntu
Name of the file on the remote system	documents.txt
Name that we wish to store the file as on our system	notes.txt

The command will now look like the following: `scp ubuntu@192.168.1.30:/home/ubuntu/documents.txt notes.txt`

## Serving Files From Your Host - WEB

Ubuntu machines come pre-packaged with python3. Python helpfully provides a lightweight and easy-to-use module called "HTTPServer". This module turns your computer into a quick and easy web server that you can use to serve your own files, where they can then be downloaded by another computing using commands such as `curl` and `wget`.

Python3's "HTTPServer" will serve the files in the directory where you run the command, but this can be changed by providing options that can be found within the manual pages. Simply, all we need to do is run `python3 -m http.server`. Below, we are serving from a directory called "webserver", which has a single named "file".

## Using Python to start a web server

```
tryhackme@linux3:/webserver# python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

Now, let's use `wget` to download the file using the 10.201.19.114 address and the name of the file. Remember, because the python3 server is running port 8000, you will need to specify this within your wget command. For example:

## An example wget command of a web server running on port 8000

```
tryhackme@mymachine:~# wget http://10.201.19.114:8000/myfile
```

Note, you will need to open a new terminal to use `wget` and leave the one that you have started the Python3 web server in. This is because, once you start the Python3 web server, it will run in that terminal until you cancel it.

## Downloading a file from our webserver using wget

```
tryhackme@linux3:/tmp# wget http://10.201.19.114:8000/file
2021-05-04 14:26:16 http://127.0.0.1:8000/file
Connecting to http://127.0.0.1:8000... connected.
HTTP request sent, awaiting response... 200 OK
```

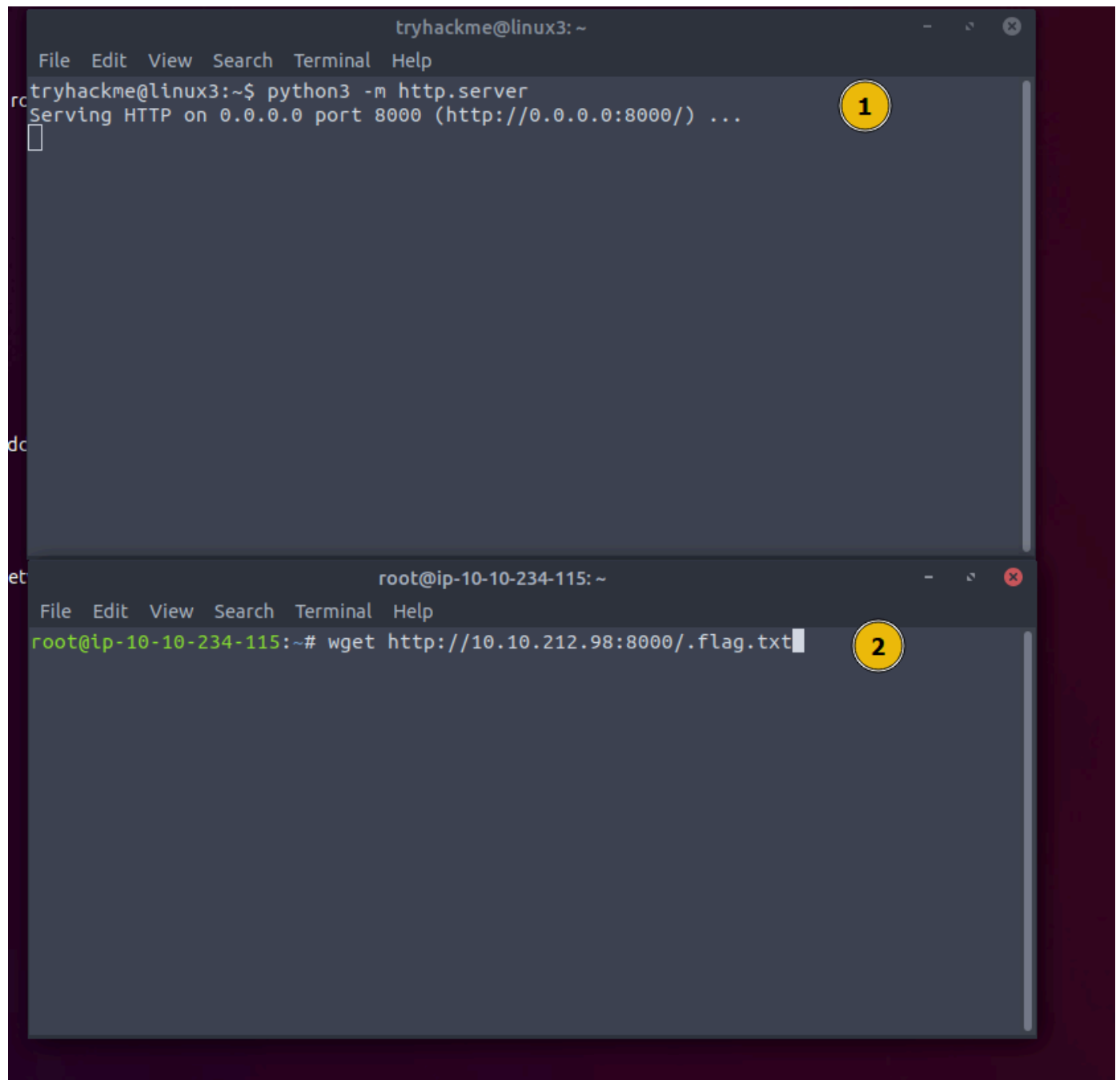
```
Length: 51095 (50K) [text]
```

```
Saving to: 'file'
```

```
file 100%[=====>]  
49.90K --.-KB/s in 0.04s
```

```
2021-05-04 14:26:16 (1.31 MB/s) - 'file' saved [51095/51095]
```

**Remember**, you will need to run the `wget` command in another terminal (while keeping the terminal that is running the Python3 server active).



The Python `http.server` module is simple but limited — you have to know the exact file name and location to serve it. Updog is a better option: a lightweight, more advanced web server that makes serving files easier.

# Processes

Processes are the programs that are running on your machine. They are managed by the kernel, where each process will have an ID associated with it, also known as its PID. The PID increments for the order in which the process starts. For example, the 60th process will have a PID of 60.

## Viewing Processes

We can use the `ps` command to provide a list of the running processes as our user's session and some additional information such as its status code, the session that is running it, how much usage time of the CPU it is using, and the name of the actual program or command that is being executed

```
cmnatic@CMNatic-THM-LPTOP:~$ ps
  PID TTY          TIME CMD
   102 pts/1        00:00:00 bash
   204 pts/1        00:00:00 ps
cmnatic@CMNatic-THM-LPTOP:~$ ps
  PID TTY          TIME CMD
   102 pts/1        00:00:00 bash
   205 pts/1        00:00:00 ps
cmnatic@CMNatic-THM-LPTOP:~$
```

Note how in the screenshot above, the second process `ps` has a PID of 204, and then in the command below it, this is then incremented to 205.

To see the processes run by other users and those that don't run from a session (i.e. system processes), we need to provide `aux` to the `ps` command like so: `ps aux`

```
cmnatic@CMNatic-THM-LPTOP:~$ ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   892    580 ?        Ss   Apr24    0:00 /init
root       100  0.0  0.0   892     84 ?        Ss   13:28    0:00 /init
root       101  0.0  0.0   892     84 ?        R    13:28    0:00 /init
cmnatic    102  0.0  0.0  10032  4984 pts/1    Ss   13:28    0:00 -bash
cmnatic    206  0.0  0.0  10616  3288 pts/1    R+   22:32    0:00 ps aux
cmnatic@CMNatic-THM-LPTOP:~$
```

Note we can see a total of 5 processes -- note how we now have "root" and "cmnatic"

Another very useful command is the `top` command; `top` gives you real-time statistics about the processes running on your system instead of a one-time view. These statistics will refresh every 10 seconds, but will also refresh when you use the arrow keys to browse the various rows. Another great command to gain insight into your system is via the `top` command

```
top - 22:36:17 up 1 day, 6:32, 0 users, load average: 0.00, 0.00, 0.00
Tasks: 5 total, 1 running, 4 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.8 sy, 0.0 ni, 99.2 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 12630.9 total, 12206.5 free, 83.6 used, 340.9 buff/cache
MiB Swap: 4096.0 total, 4096.0 free, 0.0 used. 12306.1 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	892	580	516	S	0.0	0.0	0:00.93	init
100	root	20	0	892	84	20	S	0.0	0.0	0:00.00	init
101	root	20	0	892	84	20	S	0.0	0.0	0:00.07	init
102	cmnatic	20	0	10032	4988	3272	S	0.0	0.0	0:00.08	bash
209	cmnatic	20	0	10872	3704	3188	R	0.0	0.0	0:00.00	top

## Managing Processes

You can send signals that terminate processes; there are a variety of types of signals that correlate to exactly how "cleanly" the process is dealt with by the kernel. To kill a command, we can use the appropriately named `kill` command and the associated PID that we wish to kill. i.e., to kill PID 1337, we'd use `kill 1337`.

Below are some of the signals that we can send to a process when it is killed:

- SIGTERM - Kill the process, but allow it to do some cleanup tasks beforehand
- SIGKILL - Kill the process - doesn't do any cleanup after the fact
- SIGSTOP - Stop/suspend a process

## How do Processes Start?

The Operating System (OS) uses namespaces to split up the resources available on the computer to (such as CPU, RAM and priority) processes. Think of it as splitting your computer up into slices -- similar to a cake. Processes within that slice will have access to a certain amount of computing power, however, it will be a small portion of what is actually available to every process overall.

Namespaces are great for security as it is a way of isolating processes from another -- only those that are in the same namespace will be able to see each other.

We previously talked about how PID works, and this is where it comes into play. The process with an ID of 0 is a process that is started when the system boots. This process is the system's init on Ubuntu, such as **systemd**, which is used to provide a way of managing a user's processes and sits in between the operating system and the user.

For example, once a system boots and it initializes, **systemd** is one of the first processes that are started. Any program or piece of software that we want to start will start as what's known as a child process of **systemd**. This means that it is controlled by **systemd**, but will run as its own process (although sharing the resources from **systemd**) to make it easier for us to identify.

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	101800	11340	8400	S	0.0	1.1	0:11.74	systemd

## Getting Processes/Services to Start on Boot

Some applications can be started on the boot of the system that we own. For example, web servers, database servers or file transfer servers. This software is often critical and is often told to start during the boot-up of the system by administrators.

In this example, we're going to be telling the apache web server to be starting apache manually and then telling the system to launch apache2 on boot.

Enter the use of `systemctl` -- this command allows us to interact with the **systemd** process/daemon. Continuing on with our example, `systemctl` is an easy to use command that takes the following formatting: `systemctl [option] [service]`

For example, to tell apache to start up, we'll use `systemctl start apache2`. Same with if we wanted to stop apache, we'd just replace the `[option]` with stop.

We can do four options with `systemctl`

- Start
- Stop
- Enable
- Disable

## An Introduction to Backgrounding and Foregrounding in Linux

Processes can run in two states: In the background and in the foreground. For example, commands that you run in your terminal such as "echo" or things of that sort will run in the foreground of your terminal as it is the only command provided that hasn't been told to run in the background. "Echo" is a great example as the output of echo will return to you in the foreground, but wouldn't in the background - take the screenshot below, for example.

```
root@linux3:~# echo "Hi THM"
Hi THM
root@linux3:~# echo "Hi THM" &
[1] 16889
root@linux3:~# Hi THM
```

Here we're running `echo "Hi THM"`, where we expect the output to be returned to us like it is at the start. But after adding the `&` operator to the command, we're instead just given the ID of the echo process rather than the actual output -- as it is running in the background.



This is great for commands such as copying files because it means that we can run the command in the background and continue on with whatever further commands we wish to execute (without having to wait for the file copy to finish first)

We can do the same when executing things like scripts -- rather than relying on the `&` operator, we can use `Ctrl + Z` on our keyboard to background a process. It is also an effective way of "pausing" the execution of a script or command like in the example below:

```
This will keep on looping until I stop it!  
This will keep on looping until I stop it!  
This will keep on looping until I stop it!  
This will keep on looping until I stop it!  
This will keep on looping until I stop it!  
This will keep on looping until I stop it!  
This will keep on looping until I stop it!  
This will keep on looping until I stop it!  
This will keep on looping until I stop it!  
This will keep on looping until I stop it!  
This will keep on looping until I stop it!  
This will keep on looping until I stop it!  
T^Z  
[1]+  Stopped                  ./background.sh  
root@linux3:/var/opt#
```

This script will keep on repeating "This will keep on looping until I stop!" until I stop or suspend the process. By using `Ctrl + Z` (as indicated by `T^Z`). Now our terminal is no longer filled up with messages until we foreground it.

## Foregrounding a process

Now that we have a process running in the background, for example, our script "background.sh" which can be confirmed by using the `ps aux` command, we can back-pedal and bring this process back to the foreground to interact with.

```
root      19802  0.9  0.3  8616  3108 pts/1    T   15:37   0:01 /bin/bash ./background.sh  
root      20995  0.0  0.0    0      0 ?        I   15:40   0:00 [kworker/u30:1-events_unbound]  
ubuntu    21007  0.0  0.0   7228   592 ?        S   15:40   0:00 sleep 1  
root      21008  0.0  0.3  10616  3452 pts/1    R+  15:40   0:00 ps aux  
root@linux3:/var/opt#
```

With our process backgrounded using either `Ctrl + Z` or the `&` operator, we can use `fg` to bring this back to focus like below, where we can see the `fg` command is being used to bring the background process back into use on the terminal, where the output of the script is now returned to us.

```
root@linux3:/var/opt# fg
```

```
This will keep on looping until I stop it!  
This will keep on looping until I stop it!  
This will keep on looping until I stop it!  
This will keep on looping until I stop it!  
This will keep on looping until I stop it!  
This will keep on looping until I stop it!  
This will keep on looping until I stop it!  
This will keep on looping until I stop it!  
This will keep on looping until I stop it!  
This will keep on looping until I stop it!  
This will keep on looping until I stop it!  
This will keep on looping until I stop it!  
This will keep on looping until I stop it!
```

## Maintaining Your System

---

Users may want to schedule a certain action or task to take place after the system has booted. For example, running commands, backing up files, or launching programs such as Spotify or Google Chrome.

Crontab is one of the processes that is started during boot, which is responsible for facilitating and managing cron jobs.

```
GNU nano 4.8 /tmp/crontab.OUI4VJ/crontab
# 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
```

A **crontab** is a special file with formatting that is recognized by the `cron` process to execute each line step-by-step. Crontabs require 6 specific values:

Value	Description
MIN	What minute to execute at
HOUR	What hour to execute at
DOM	What day of the month to execute at
MON	What month of the year to execute at
DOW	What day of the week to execute at
CMD	The actual command that will be executed.

Let's use the example of backing up files. You may wish to backup "cmnatic"'s "Documents" every 12 hours.

```
0 */12 * * * cp -R /home/cmnatic/Documents /var/backups/
```

An asterisk (\*) in a crontab field means 'run no matter what' meaning we don't care about that value, just the schedule we set.

Crontabs can be edited by using `crontab -e`, where you can select an editor (such as Nano) to edit your crontab.

Cron Job Generated (you may copy & paste it to your crontab):

```
0 */12 * * * cp -R /home/cmnatic/Documents /var/backups/ >/dev/null 2>&1
```

Your cron job will be run at: (5 times displayed)

- 2021-04-26 00:00:00 UTC
- 2021-04-26 12:00:00 UTC
- 2021-04-27 00:00:00 UTC
- 2021-04-27 12:00:00 UTC
- 2021-04-28 00:00:00 UTC
- ...

```
GNU nano 4.8 /tmp/crontab.0UI4VJ/crontab
# 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
0 */12 * * * cp -R /home/cmnatic/Documents /var/backups/ >/dev/null 2>&1
```

## Introducing Packages & Software Repos

When developers submit software to the community, they will submit it to an "apt" repository. If approved, their programs and tools will be released.

When using the `ls` command on a Ubuntu 20.04 Linux machine, these files serve as the gateway/registry.

```
ubuntu@ip-10-10-29-121:/etc/apt$ ls
apt.conf.d  auth.conf.d  preferences.d  sources.list  sources.list.d  trusted.gpg.d
ubuntu@ip-10-10-29-121:/etc/apt$
```

## Managing Your Repositories (Adding and Removing)

Normally we use the `apt` command to install software onto our Ubuntu system. The `apt` command is a part of the package management software also named `apt`. `apt` contains a whole suite of tools that allows us to manage the packages and sources of our software, and to install or remove software at the same time.

You can install software through the use of package installers such as `dpkg`, the benefits of `apt` means that whenever we update our system -- the repository that contains the pieces of software that we add also gets checked for updates.

When adding software, the integrity of what we download is guaranteed by the use of what is called GPG (Gnu Privacy Guard) keys. These keys are essentially a safety check from the developers saying, "here's our software". If the keys do not match up to what your system trusts and what the developers used, then the software will not be downloaded.

## Maintaining Your System: Logs

---

These are some some logs from three services running on a Ubuntu machine

- An Apache2 web server
- Logs for the fail2ban service, which is used to monitor attempted brute forces, for example
- The UFW service which is used as a firewall

```

ubuntu@ip-172-31-23-158:/var/log$ ls
alternatives.log      dpkg.log              lastlog
alternatives.log.1    dpkg.log.1            letsencrypt
alternatives.log.2.gz dpkg.log.2.gz          lxd
alternatives.log.3.gz dpkg.log.3.gz          mysql
alternatives.log.4.gz dpkg.log.4.gz          syslog
alternatives.log.5.gz dpkg.log.5.gz          syslog.1
alternatives.log.6.gz dpkg.log.6.gz          syslog.2.gz
alternatives.log.7.gz dpkg.log.7.gz          syslog.3.gz
amazon                dpkg.log.8.gz          syslog.4.gz
apache2               dpkg.log.9.gz          syslog.5.gz
apport.log            fail2ban.log           syslog.6.gz
apport.log.1          fail2ban.log.1         syslog.7.gz
apt                  fail2ban.log.2.gz      tallylog
auth.log              fail2ban.log.3.gz      ufw.log
auth.log.1            fail2ban.log.4.gz      ufw.log.1
auth.log.2.gz         fontconfig.log         ufw.log.2.gz
auth.log.3.gz         journal                ufw.log.3.gz
auth.log.4.gz         kern.log               ufw.log.4.gz
btmtp                 kern.log.1             unattended-upgrades
btmtp.1               kern.log.2.gz          wtmp
cloud-init-output.log kern.log.3.gz           wtmp.1
cloud-init.log         kern.log.4.gz
dist-upgrade          landscape
ubuntu@ip-172-31-23-158:/var/log$

```

These services and logs are a way to monitor your system and protect it.

The logs for services such as a web server contain information about every single request - allowing developers or administrators to diagnose performance issues or investigate an intruder's activity.

- access log
- error log

```

ubuntu@ip-172-31-23-158:/var/log/apache2$ ls
access.log      access.log.3.gz  error.log.1      error.log.4.gz
access.log.1    access.log.4.gz  error.log.10.gz  error.log.5.gz
access.log.10.gz access.log.5.gz  error.log.11.gz  error.log.6.gz
access.log.11.gz access.log.6.gz  error.log.12.gz  error.log.7.gz
access.log.12.gz access.log.7.gz  error.log.13.gz  error.log.8.gz
access.log.13.gz access.log.8.gz  error.log.14.gz  error.log.9.gz
access.log.14.gz access.log.9.gz  error.log.2.gz   other_vhosts_access.log
access.log.2.gz error.log         error.log.3.gz
ubuntu@ip-172-31-23-158:/var/log/apache2$

```

There are logs that store information about how the OS is running itself and actions that are performed by users, such as authentication attempts.