[CentOS-announce] CEBA-2021:0863 CentOS 7 lvm2 BugFix Update

CentOS Errata and Bugfix Advisory 2021:0863
Upstream details at : https://access.redhat.com/errata/RHBA-2021:0863

The following updated files have been uploaded and are currently

syncing to the mirrors: (sha256sum Filename)

x86_64:

71f20a3e8a0bfe010564d4b6ff77d4b2647ba8383e62e64a557c131359fff889 cmirror-2.02.187-6.el7_9.4.x86_64.rpm

2ed2fcd8e79d8a62a374c054083792b657cb4d4668e39188deeb1ad3d72c948f device-mapper-1.02.170-6.el7_9.4.x86_64.rpm

88e796afa3934d3cefb09a6a575714bdc8aede8ee6a3c6b7be0bb59a130d5efd device-mapper-devel-1.02.170-6.el7_9.4.i686.rpm

39c70285432c56fe5dd79f0d155069d48956d10fcdcac838a4590c4a44b0b521 device-mapper-devel-1.02.170-6.el7_9.4.x86_64.rpm

96abc02747879aa68bd335dadad5f1e2d5ea64c519db929820194757e80af933 device-mapper-event-1.02.170-6.el7_9.4.x86_64.rpm

3feced91f7391352cfca0debae0c2aa43febbeee7d24929c75cd8d1d5a6b782e device-mapper-event-devel-1.02.170-6.el7_9.4.i686.rpm

0986dce8e3a59471cabc7ed457e23c3f0fdc81e3cbab4093eaa0426e2d5d26ef device-mapper-event-devel-1.02.170-6.el7_9.4.x86_64.rpm

```
12fb546150a8175c1aa95419a75e512972cb4c6c8eea1ade84e3319b1cfada
ea  device-mapper-event-libs-1.02.170-6.el7_9.4.i686.rpm

68e1429d8531b268902804fdf15f6eb8ffcadc2d36315c6090a3edb14eb2ab
13  device-mapper-event-libs-1.02.170-6.el7_9.4.x86_64.rpm

85c0d75554f340d99942609a8488c0ad5b2e4ddc0e40c1192056ab8ab83954
92  device-mapper-libs-1.02.170-6.el7_9.4.i686.rpm

40ebdc19d1f43fa0892059019bcdba4bd97351c6d146ef74e530e00dc4732c
4c  device-mapper-libs-1.02.170-6.el7_9.4.x86_64.rpm

9d34ca3784c0b623e48bc95dca03187a3404a69dd41baa19138db561f25be0
f9  lvm2-2.02.187-6.el7_9.4.x86_64.rpm

14b4703549cb70dba8871814b76da451a63a067a0c48cbf78c90a94b528e80
71  lvm2-cluster-2.02.187-6.el7_9.4.x86_64.rpm

75f984b4cdee8c054416401d8e1a4f00cb8d0fa104e54d12456f8884c58d7d
b5  lvm2-devel-2.02.187-6.el7_9.4.i686.rpm

06d86e24f67250850372882170efd7d12d041a90b28a72fb952a0f3d101418
25  lvm2-devel-2.02.187-6.el7_9.4.x86_64.rpm

bd945872965d8eeb4accc4fc087400341b7ecfd21918499fa21ff68810cb41
94  lvm2-libs-2.02.187-6.el7_9.4.i686.rpm

fdbf4bfe28188e74649c64f6a210cdce1aec37dafe82324cc4a7403e012e4c
4f  lvm2-libs-2.02.187-6.el7_9.4.x86_64.rpm

a1e667c441a340c5ac1f0544c4a3d934fde4bf968688af42f5d33e773f092b
5b  lvm2-lockd-2.02.187-6.el7_9.4.x86_64.rpm

bff2adfbc894a60df24055ab573b81c8128737c5d8b2530955f48d64f1c121
b5  lvm2-python-boom-1.2-2.el7_9.4.noarch.rpm

c3b133e5f06890f86c2c918c35177dfe1000500cfee3fc5ff248faab874324
```

f1 lvm2-python-libs-2.02.187-6.el7_9.4.x86_64.rpm

76398082a1500552d12445932d6e5689e02b1ebdced0a38e4003a7fa56871c
e1 lvm2-sysvinit-2.02.187-6.el7_9.4.x86_64.rpm

Source:

e524f3b8629d666bb07fca6db6c4cf645fade58651dee4ccbe851534259adf
17 lvm2-2.02.187-6.el7_9.4.src.rpm

_

Johnny Hughes

CentOS Project { http://www.centos.org/ }

irc: hughesjr, #[hidden email]

Twitter: @JohnnyCentOS

____

CentOS-announce mailing list

[hidden email]

https://lists.centos.org/mailman/listinfo/centos-announce

How To: Time a Command

Have you ever wanted to know how long it takes to complete a command that you entered in the Linux terminal? Well, wonder no more!

This is going to just be a pretty quick article and easy to follow. There’s not a whole lot to explain and it’s pretty straightforward. Like often, let’s crack open your default terminal by pressing CTRL + ALT + T on your keyboard.

Now, let’s take the command:

```
[code]ls -la[/code]
```

Unless you have a lot of files, that completes pretty quickly. But, how fast does it really take? Well, simply add the ‘time‘ command before it. Time is simply described in the man page as:

> *time – run programs and summarize system resource usage*

And, for today, it’s going to be pretty easy to use that command. To find out how long it took to list all the files and folders in a directory, you could use:

```
[code]time ls -la[/code]
```

Note how it tells you the time beneath the results and, if you want to try something bigger, you can take a look at this command:

```
[code]for i in {0..99999}; do echo “I love LinuxTips!”; done[/code]
```

That should take a just a little more time, but you can actually see how long it really took by adding ‘time’ in front of it. So:

```
[code]time for i in {0..99999}; do echo “I love LinuxTips!”; done[/code]
```

The output at the end is something like this:

```
real 0m0.566s
user 0m0.423s
```

```
  sys 0m0.143s
```

The ‘real’ is how much time it really took. The ‘user’ is how much time it took for the user. The ‘sys’ is how much time it took for the system – the amount of time that the kernel actually devoted to it.

So, there you have it! You can use the time command to find out how long it takes to run stuff in your terminal. If you’re playing with scripting and you’re looking to optimize it, this is a valuable tool. If you’re just a bit curious, then you now have a new tool.

I told you that it’d be quick and easy! Like always, thanks for reading. Feel free to sign up for the newsletter. I promise to not spam you or sell your email address.

USN-4881-1: containerd vulnerability

It was discovered that containerd incorrectly handled certain environment variables. Contrary to expectations, a container could receive environment variables defined for a different container, possibly containing sensitive information.

Generating a Software Bill of Materials (SBOM) with Open Source Standards and Tooling

Every month there seems to be a new software vulnerability showing up on social media, which causes open source program offices and security teams to start querying their inventories to see how FOSS components they use may impact their organizations.

Frequently this information is not available in a consistent format within an organization for automatic querying and may result in a significant amount of email and manual effort. By exchanging software metadata in a standardized software bill of materials (SBOM) format between organizations, automation within an organization becomes simpler, accelerating the discovery process and uncovering risk so that mitigations can be considered quickly.

In the last year, we’ve also seen standards like OpenChain (ISO/IEC 5320:2020) gain adoption in the supply chain. Customers have started asking for a bill of materials from their suppliers as part of negotiation and contract discussions to conform to the standard. OpenChain has a focus on ensuring that there is sufficient information for license compliance, and as a result, expects metadata for the distributed components as well. A software bill of materials can be used to support the systematic review and approval of each component’s license terms to clarify the obligations and restrictions as it applies to the distribution of the supplied software and reduces risk.

Kate Stewart, VP, Dependable Embedded Systems, The Linux Foundation, will host a complimentary mentorship webinar entitled **Generating Software Bill Of Materials** on Thursday, March 25 at 7:30 am PST. This session will work through the minimum elements included in a software bill of materials and detail the reasoning behind why those elements are included. To register, please click here.

Register for webinar
There are many ways this software metadata can be shared. The common SBOM document format options (SPDX, SWID, and CycloneDX) will be reviewed so that the participants can better understand what is available for those just starting.

This mentorship session will work through some simple examples and then guide where to find the next level of details and further references.

At the end of this session, participants will be on a secure footing and a path towards the automated generation of SBOMs as part of their build and release processes in the future.

The post Generating a Software Bill of Materials (SBOM) with Open Source Standards and Tooling appeared first on Linux Foundation.

USN-4880-1: OpenJPEG vulnerabilities

It was discovered that OpenJPEG incorrectly handled certain image data. An
attacker could use this issue to cause OpenJPEG to crash, leading to a
denial of service, or possibly execute arbitrary code.