Upgrading RHEL 8 to RHEL 8.1

Red Hat Enterprise Linux 8

Needed to reboot my **Red Hat Enterprise Linux 8** desktop anyway, so decided to upgrade it to RHEL 8.1.

Check That Your Software Subscription is Active

For example, I realised that I have still been using the RHEL 8 beta subscription instead of the Developers License. After **completing Red Hat subscription registration**, I got the following:

```
greys@redhat:~ $ sudo subscription-manager list
+-------------------------------------------+
Installed Product Status
+-------------------------------------------+
Product Name:   Red Hat Enterprise Linux for x86_64
Product ID:     479
Version:        8.1
Arch:           x86_64
```

```
Status:         Subscribed
Status Details:
Starts:         14/11/19
Ends:           13/11/20
```

Upgrade Red Hat OS with yum-update

yum tools are more integrated that many people though! So I’m still using **yum update** instead of **dnf**:

```
root@redhat:~ # yum update
 Updating Subscription Management repositories.
 Red Hat Enterprise Linux 8 for x86_64 - AppStream (RPMs)
2.6 kB/s | 4.5 kB     00:01
 Red Hat Enterprise Linux 8 for x86_64 - AppStream (RPMs)
3.0 MB/s |  13 MB     00:04
 Red Hat Enterprise Linux 8 for x86_64 - BaseOS (RPMs)
2.4 kB/s | 4.1 kB     00:01
 Red Hat Enterprise Linux 8 for x86_64 - BaseOS (RPMs)
2.4 MB/s |  12 MB     00:04
 Last metadata expiration check: 0:00:01 ago on Thu 14 Nov
2019 21:28:59 GMT.
 Dependencies resolved.
  Package                                              Arch
Version                                          Repository
Size
 Installing:
  yum-utils                                            noarch
4.0.8-3.el8                                      rhel-8-for-
x86_64-baseos-rpms        64 k
      replacing  dnf-utils.noarch 4.0.2.2-3.el8
  kernel-debug-devel                                   x86_64
4.18.0-147.0.3.el8_1                             rhel-8-for-
x86_64-baseos-rpms        14 M
  kernel-devel                                         x86_64
4.18.0-147.0.3.el8_1                             rhel-8-for-
x86_64-baseos-rpms        13 M
```

```
  kernel-core                                                  x86_64
4.18.0-147.0.3.el8_1                                         rhel-8-for-
x86_64-baseos-rpms        25 M
  kernel                                                       x86_64
4.18.0-147.0.3.el8_1                                         rhel-8-for-
x86_64-baseos-rpms       1.5 M
  kernel-modules                                               x86_64
4.18.0-147.0.3.el8_1                                         rhel-8-for-
x86_64-baseos-rpms        22 M
 Upgrading:
  netcf-libs                                                   x86_64
0.2.8-12.module+el8.1.0+4066+0f1aadab                        rhel-8-for-
x86_64-appstream-rpms     77 k
  libXt                                                        x86_64
1.1.5-12.el8                                                 rhel-8-for-
x86_64-appstream-rpms    185 k
  alsa-utils                                                   x86_64
1.1.9-1.el8                                                  rhel-8-for-
x86_64-appstream-rpms    1.1 M
...
  podman-manpages                                              noarch
1.4.2-5.module+el8.1.0+4240+893c1ab8                         rhel-8-for-
x86_64-appstream-rpms    180 k
  python3-pip-wheel                                            noarch
9.0.3-15.el8                                                 rhel-8-for-
x86_64-baseos-rpms       1.2 M
  mozjs60                                                      x86_64
60.9.0-3.el8                                                 rhel-8-for-
x86_64-baseos-rpms       6.7 M
  libssh-config                                                noarch
0.9.0-4.el8                                                  rhel-8-for-
x86_64-baseos-rpms        18 k
  python3-setuptools-wheel                                     noarch
39.2.0-5.el8                                                 rhel-8-for-
x86_64-baseos-rpms       289 k
 Installing weak dependencies:
  oddjob-mkhomedir                                             x86_64
0.34.4-7.el8                                                 rhel-8-for-
x86_64-appstream-rpms     52 k
  libvarlink                                                   x86_64
18-3.el8                                                     rhel-8-for-
```

```
x86_64-baseos-rpms        44 k
 Transaction Summary
 Install   17 Packages
 Upgrade  646 Packages
 Total download size: 1.2 G
 Is this ok [y/N]:
```

Some 15min later I had my desktop in a much better shape:

```
...
   sos-3.7-6.el8_1.noarch
   hwdata-0.314-8.2.el8_1.noarch
   ca-certificates-2019.2.32-80.0.el8_1.noarch
   microcode_ctl-4:20190618-1.20191112.1.el8_1.x86_64
   kernel-tools-4.18.0-147.0.3.el8_1.x86_64
   kernel-headers-4.18.0-147.0.3.el8_1.x86_64
   bpftool-4.18.0-147.0.3.el8_1.x86_64
   kernel-tools-libs-4.18.0-147.0.3.el8_1.x86_64
   python3-perf-4.18.0-147.0.3.el8_1.x86_64
 Installed:
                          yum-utils-4.0.8-3.el8.noarch
kernel-debug-devel-4.18.0-147.0.3.el8_1.x86_64
             kernel-devel-4.18.0-147.0.3.el8_1.x86_64
kernel-core-4.18.0-147.0.3.el8_1.x86_64
                   kernel-4.18.0-147.0.3.el8_1.x86_64
kernel-modules-4.18.0-147.0.3.el8_1.x86_64
                 oddjob-mkhomedir-0.34.4-7.el8.x86_64
libvarlink-18-3.el8.x86_64
               python3-argcomplete-1.9.3-6.el8.noarch
oddjob-0.34.4-7.el8.x86_64
                                  tbb-2018.2-9.el8.x86_64
gnome-shell-extension-horizontal-
workspaces-3.32.1-10.el8.noarch
   podman-manpages-1.4.2-5.module+el8.1.0+4240+893c1ab8.noarch
python3-pip-wheel-9.0.3-15.el8.noarch
                             mozjs60-60.9.0-3.el8.x86_64
libssh-config-0.9.0-4.el8.noarch
   python3-setuptools-wheel-39.2.0-5.el8.noarch
 Complete!
```

```
root@redhat:~ #
```

And that’s it! I rebooted the server and my OS is **RHEL 8.1** now:

```
greys@redhat:~ $ more /etc/redhat-release
Red Hat Enterprise Linux release 8.1 (Ootpa)
greys@redhat:~ $ uname -a
Linux redhat 4.18.0-147.0.3.el8_1.x86_64 #1 SMP Mon Nov 11
12:58:36 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
```

See Also

- Red Hat Linux
- RHEL 8 Reference
- screenFetch in RHEL 8
- Red Hat Enterprise Linux 8.1 released
- **Linux Commands**

Red Hat Enterprise Linux 8.1

RHEL 8

Just as I published last Unix Tutorial Digest on November 5th, **RHEL 8.1** release got shipped – think this is a great incremental release bringing a number of key improvements to the **Red Hat Enterprise Linux 8**.

RHEL 8 Release Cadence

Red Hat announced that going forward **Red Hat Enterprise Linux OS** will be receiving regular updates every 6 months. Since RHEL 8 release was in May 2019, this current RHEL 8.1 update is right on time, 6 months after.

RHEL 8.1 Improvements I Want To Try

There's a number of great improvements in this release:

- **Live Kernel Patching with kpatch**
- **SELinux** profiles for containers and tbolt for Thunderbolt devices – will be cool to try on my RHEL 8

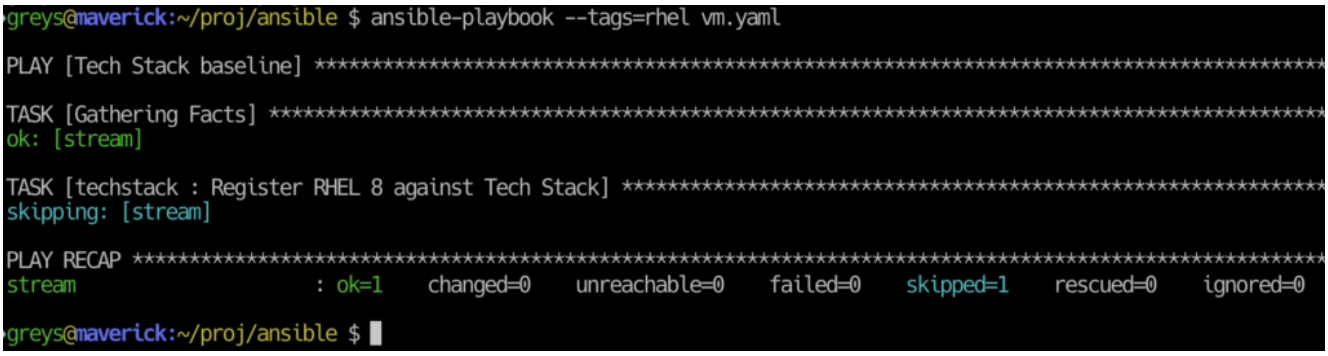
PC
- Perhaps try **RHEL 7.6 in-place upgrade to RHEL 8.1**
- Review **rhel-system-roles** and specifically the new **storage** role added in RHEL 8.1
- **LUKS2 online re-encryption**
- **RHEL 8 Web Console**
    - firewall zones management
    - Virtual Machines configuration

I also want to try **Red Hat Universal Base Image for RHEL 8** – it's been around since initial release in May, I just never got the chance to have a look.

See Also

- Red Hat Enterprise Linux 8.1 release notes
- RHEL 8 Reference
- Red Hat Linux
- Hello, world with podman
- Docker software

Run Ansible Tasks Based on OS Distribution



Skipping Tasks Based on Ansible Conditionals

I'm actively refreshing my **Ansible** setup for both servers and desktops, running mostly Red Hat Enterprise Linux and CentOS Linux. Today's quick tip is about the functionality that **Ansible** has for precise control of configuration management in such closely related distros.

How To Run Ansible Task In Specific OS Distribution

Ansible has quite a few facts it collects about each managed system, they are usually established (collected) at the very start of running any playbook (unless you decide to skip gathering facts).

A whole group of **Ansible** facts talks about OS distribution. Here they are as confirmed form the freshly deployed CentOS 8 Stream VM:

One of these facts is **ansible_distribution**:

greys@maverick:~/proj/ansible $ **ansible stream -m setup| grep distribution**

```
        "ansible_distribution": "CentOS",
"ansible_distribution_file_parsed": true,
"ansible_distribution_file_path": "/etc/redhat-release",
"ansible_distribution_file_variety": "RedHat",
"ansible_distribution_major_version": "8",
"ansible_distribution_release": "Core",
"ansible_distribution_version": "8.0",
```

We can use the very first one, called ansible_distribution. For CentOS, it says “CentOS”, but for Red Hat Enterprise Linux, it will be “RedHat”.

Example of using ansible_distribution

I have the following task below. It’s activating RHEL 8 subscription using my account, but obviously should only be doing this for Red Hat systems. For **CentOS**, I simply want to skip this task.

That’s why I’m checking for ansible_distribution, and as per below – the code will only run if and when the distribution is speficially **RedHat**, and not **CentOS** as my “**stream**” VM:

```
- name: Register RHEL 8 against Tech Stack
  shell: "subscription-manager register --activationkey=rhel8
--org=100XXXXX --force"
  tags:
    - rhel
  when:
    ansible_distribution == "RedHat"
```

That’s it! Even if I specify the **tags=rhel** filter, **ansible-playbook** will skip this task based on the collected facts (**stream** is the VM hostname):

```
greys@maverick:~/proj/ansible $ ansible-playbook --tags=rhel vm.yaml
PLAY [Tech Stack baseline] ***
TASK [Gathering Facts] *
ok: [stream]
TASK [techstack : Register RHEL 8 against Tech Stack]
skipping: [stream]
PLAY RECAP *
stream                     : ok=1    changed=0
unreachable=0    failed=0    skipped=1    rescued=0
ignored=0
```

See Also

- Ansible
- Getting Started with Ansible
- RHEL8
- Ansible 2.0
- Create a Unix group with Ansible
- Specify User per task in Ansible
- Running Ansible scripts without making changes

CentOS 8 and CentOS Stream

Released

Great news, **CentOS 8 is released** now. Even better – there's now a step in between Fedora and RHEL, called CentOS Stream.

Have you tried them yet? I'll be **upgrading to CentOS 8** this week and am also thinking of downloading and installing CentOS Stream in a KVM VM.

Let me know what you think!

See Also

- CentOS
- Download CentOS Linux
- RHEL 8
- Red Hat Linux
- Upgrading to CentOS 7.7

Attach Interface to Specific Firewall Zone in RHEL 8

RHEL 8

One of the first things I had to do on my recently built RHEL 8 PC was to move the primary network interface from public (default) zone to home zone – to make sure any firewall ports I open stay private enough.

How To List Which Zones and Interfaces are Active

Using the **get-active-zones** option of the firewall-cmd command, it's possible to confirm where **eno1** interface is at the moment. It's already in the **home** zone cause I made the update earlier:

```
root@redhat:~ # firewall-cmd --get-active-zones
home
  interfaces: eno1
libvirt
  interfaces: virbr0
```

Attach Interface to a Firewall Zone

Here’s how one can move specified interface into a zone we want:

```
root@redhat:~ # firewall-cmd --zone=home --change-interface=eno1
success
```

Just to show how it works, I’m going to move eno1 into public zone and back to home one:

```
root@redhat:~ # firewall-cmd --zone=public --change-interface=eno1
success
root@redhat:~ # firewall-cmd --get-active-zones
libvirt
  interfaces: virbr0
public
  interfaces: eno1
```

Making Sure Firewall Changes Are

Permanent

Don’t forget that after confirming a working firewall configuration, you need to re-run the same command with **permanent** option – this will update necessary files to make sure your firewall changes can survive a reboot:

```
root@redhat:~ # firewall-cmd --zone=home --change-interface=eno1 --permanent
The interface is under control of NetworkManager, setting zone to 'home'.
success
```

That’s it for today. Am really enjoying **RHEL 8** configuration and still have this feeling I barely scratch the surface with all the new improvements this Red Hat Enterprise Linux brings.

See Also

- Red Hat Linux
- RHEL 8 Reference
- Confirm firewall configuration in RHEL 8
- Advanced Unix Commands
- Linux Commands

Hello, World in podman

RHEL 8

Turns out it’s not that easy to install Docker CE in **RHEL 8** just yet. Well, maybe there’s no immediate need since RHEL 8 comes with its own containerization stack based on **podman**?

Hello, World in podman

podman provides comprehensive compatibility with **docker** command, most non-Docker specific options are supported.

If you are familiar with **docker** command syntax, give it a try by just replacing docker with **podman** command.

Let’s do the hello world exercise:

```
greys@redhat:~ $ podman run hello-world
Trying to pull registry.redhat.io/hello-world:latest…Failed
Trying to pull quay.io/hello-world:latest…Failed
Trying to pull docker.io/hello-world:latest…Getting image source signatures
Copying blob 1b930d010525: 977 B / 977 B [==========================================] 0s
Copying config fce289e99eb9: 1.47 KiB / 1.47 KiB [==========================] 0s
Writing manifest to image destination
Storing signatures

Hello from Docker!

This message shows that your installation appears to be working correctly.



To try something more ambitious, you can run an Ubuntu container with:

$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:

https://hub.docker.com/

For more examples and ideas, visit:

https://docs.docker.com/get-started/
```

As you can see, **podman** searches in Red Hat and Quay image repositories before moving on to Docker registry, but finally gets the hello-world image there.

Run Ubuntu image in podman

And if we want to follow Docker’s advice and try running the Ubuntu Docker image, we’ll replace

docker run -it ubuntu bash

with

podman run -it ubuntu bash

… It just works:

```
greys@redhat:~ $ podman run -it ubuntu bash
Trying to pull registry.redhat.io/ubuntu:latest…Failed
Trying to pull quay.io/ubuntu:latest…Failed
Trying to pull docker.io/ubuntu:latest…Getting image source signatures
Copying blob 5667fdb72017: 25.45 MiB / 25.45 MiB [============================] 3s
Copying blob d83811f270d5: 34.53 KiB / 34.53 KiB [============================] 3s
Copying blob ee671aafb583: 850 B / 850 B [==================================] 3s
Copying blob 7fc152dfb3a6: 163 B / 163 B [==================================] 3s
Copying config 2ca708c1c9cc: 3.33 KiB / 3.33 KiB [============================] 0s
Writing manifest to image destination
Storing signatures
root@686f0d85b4ad:/# uname -a
Linux 686f0d85b4ad 4.18.0-80.el8.x86_64 #1 SMP Wed Mar 13 12:02:46 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
```

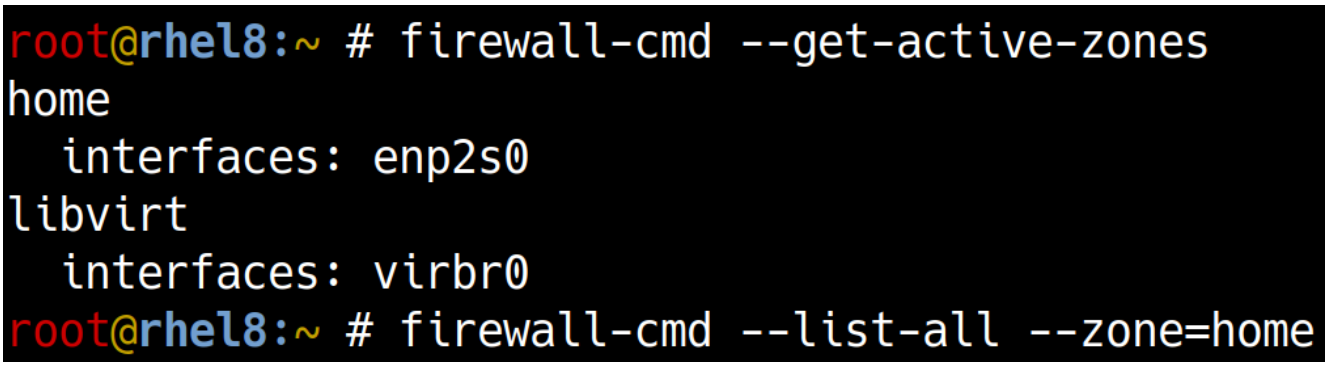
```
root@686f0d85b4ad:/# cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=18.04
DISTRIB_CODENAME=bionic
DISTRIB_DESCRIPTION="Ubuntu 18.04.3 LTS"
```

I think it's pretty cool. Will definitely read up and post more about **podman** and containerization in **Red Hat** in the following weeks.

See Also

- Docker Stop All Containers
- Docker – Restart Containers
- Docker Software

Confirm Firewall Configuration in RHEL 8



List Firewall Rules in RHEL 8

I'm fascinated by the improvements and new features in RHEL 8, plus it's a primary distro used in most corporate environments – so expect to quite a number of posts related to it in the nearest future.

The default interface for managing firewalls in RHEL 8 is **firewalld** and specifically **firewall-cmd command**.

Show Active Zones in RHEL 8

There's a concept of zones – security domains – in **RHEL 8** firewalls. You assign each of available network interfaces on your Red Hat Enterprise Linux system to one or more of these domains.

That's why the first step is to confirm these zones, to see which ones are actively managing access for each network device:

```
root@rhel8:~ # firewall-cmd --get-active-zones
home
interfaces: enp2s0
libvirt
interfaces: virbr0
```

List All Rules for Firewall Zone in RHEL 8

I’m interested in the primary physical network interface – enp2s0. It belongs to the home zone as per previous command, so that’s the zone we’ll list all the rules fore:

```
root@rhel8:~ # firewall-cmd --list-all --zone=home
home (active)
  target: default
  icmp-block-inversion: no
  interfaces: enp2s0
  sources:
  services: cockpit dhcpv6-client mdns samba-client ssh
  ports: 5901/tcp
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

From the output below, I have highlighted additionally enabled ports – 5901 is the one for VNC client that allows me to access graphics desktop session on my **RHEL 8** PC remotely.

That’s it for today! Thanks for stopping by and talk soon!

See Also

- RHEL 8
- Red Hat Enterprise Linux
- Python 3 path in RHEL 8
- Red Hat Enterprise Linux 8 – screenFetch

Skip Gathering Facts in Ansible

Red Hat Ansible

There are **Ansible playbooks** which depend on the most up-to-date information found on each node. That's where fact gathering is a much needed help. But there are also simpler more predefined playbooks, which don't need fact gathering and can therefore gain performance if no facts are collected.

Why Fact Gathering in Ansible Takes Time

Fact gathering means **Ansible** runs a number of commands to confirm the most recent values for important indicators and parameters.

Run against my freshly installed **RHEL 8** based PC, this takes roughly 4 seconds. Part of this can be to how RHEL is configured (and that it’s still a work in progress), but part of this amount of time is defined by the sheer number of facts: more than 1000!

Typical Facts Collected By Ansible

This is not a complete list, I’m just giving you examples to indicate why facts collection may be time consuming:

- hardware parameters of remote system
- storage devices (types, models, sizes, capabilities)
- filesystems and logical volume managers (objects, types, sizes)
- OS distro information
- network devices and full list of their capabilities
- environment variables

Disable Fact Gathering in Ansible

Since I don’t really need to re-establish hardware specs or logical volumes layout of my RHEL 8 desktop every time I run some Ansible post-configuration, I decided to disable fact gathering and shave 4-5 sec at the start of each playbook run.

Simply specify this at the top of your **Ansible playbook**:

gather_facts: no

In on of my playbooks, this is how it looks:

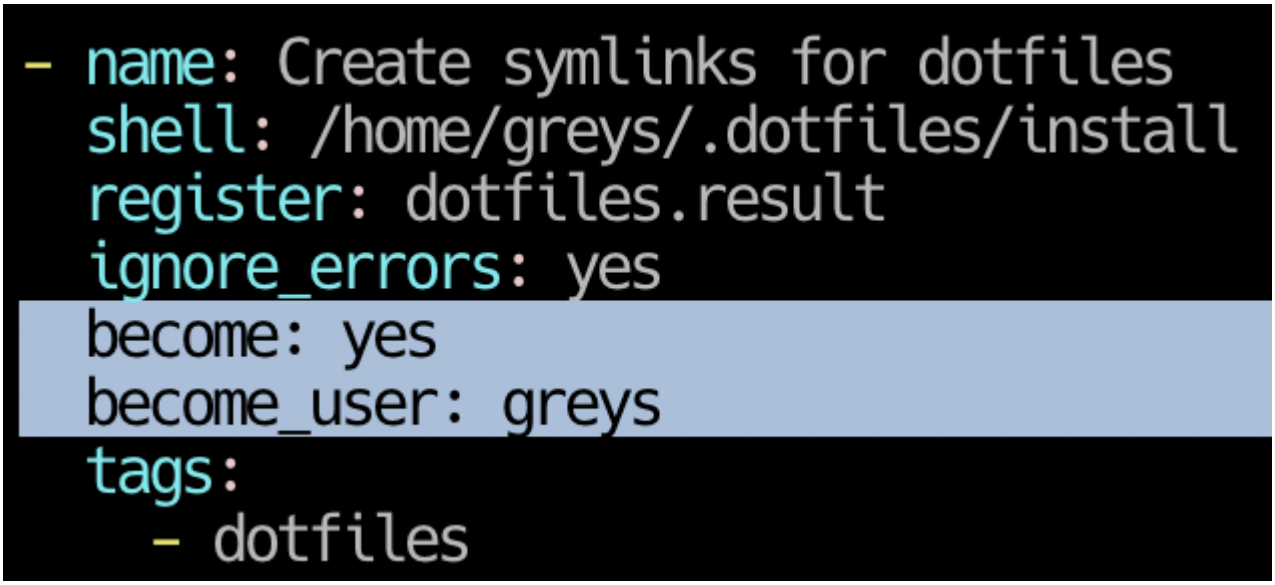
```
---
- name: Baseline
  hosts: desktops
  gather_facts: no
```

This really made a noticeable difference. Have fun!

See Also

- Ansible
- Getting started with Ansible
- Create a Unix group with Ansible
- Specify User per Task in Ansible

Specify User per Task in Ansible



become_user per task in Ansible

Turns out, **become_user** directive can be used not only for privilege escalation (running **Ansible** playbooks as root), but also for becoming any other when you want certain tasks run as that user instead of root.

Default Ansible Behavior for Running Tasks

I had the following piece of code, running **/home/greys/.dotfiles/install** script. It didn't run as intended, creating symlinks in /root directory (because that's what Ansible was running the task as):

```
- name: Create symlinks for dotfiles
  shell: /home/greys/.dotfiles/install
  register: dotfiles.result
  ignore_errors: yes
  tags:
    - dotfiles
```

Specify User for an Ansible Task

become_user parameter can be specifed per task or per playbook, apparently. So that's how you specify it per task – in my example to run the **Create symlinks for dotfiles** task as my user **greys**:

```
- name: Create symlinks for dotfiles
  shell: /home/greys/.dotfiles/install
  register: dotfiles.result
  ignore_errors: yes
  become: yes
  become_user: greys
  tags:
    - dotfiles
```

See Also

- **Ansible**
- How To Create a Unix Group with Ansible
- Dry Running Ansible Scripts
- Getting Started with Ansible

How To Troubleshoot SELinux

with Audit Logs


Audit Logs with SELinux Messags

I'm post configuring a new RHEL 8 setup on my old PC and want to share some useful **SELinux** troubleshooting techniques.

How To Check Audit Logs for SELinux

I had a problem with **SSH** not accepting keys for login. Specifically, I wanted the keys to be in a non-standard /var/ssh/greys/authorized_keys location (instead of my homedir), but SSH daemon would just ignore this file.

I double checked permissions, restarted **SSHd** and eventuall realised that the issue must have been due to SELinux. So I went to inspect the audit logs.

Red Hat Enterprise Linux puts audit logs into /var/log/audit directory. If you're looking for SELinux issues, just grep for **denied** – it will show you everything that has recently been blocked:

```
root@rhel8:~ # grep denied /var/log/audit/*
type=AVC msg=audit(1567799177.932:3031): avc:  denied  { read } for  pid=24527 comm="sshd" name="authorized_keys" dev="dm-11" ino=26047253 scontext=system_u:system_r:sshd_t:s0-s0:c0.c1023 tcontext=system_u:object_r:var_t:s0 tclass=file
```

```
permissive=0
 type=AVC msg=audit(1567799177.943:3033): avc:  denied  { read } for  pid=24527 comm="sshd" name="authorized_keys" dev="dm-11" ino=26047253 scontext=system_u:system_r:sshd_t:s0-s0:c0.c1023 tcontext=system_u:object_r:var_t:s0 tclass=file permissive=0
 type=AVC msg=audit(1567799177.956:3035): avc:  denied  { read } for  pid=24527 comm="sshd" name="authorized_keys" dev="dm-11" ino=26047253 scontext=system_u:system_r:sshd_t:s0-s0:c0.c1023 tcontext=system_u:object_r:var_t:s0 tclass=file permissive=0
```

I also highlighted the likely problem: SSH daemon is running under **sshd_t** context, but files in **/var/ssh/** directories inherited standard **var_t** context.

Just to be sure, I checked the context on the default **/home/greys/.ssh/authorized_keys** file:

```
root@rhel8:~ # ls -alZ /home/greys/.ssh/authorized_keys
 -rw-------. 1 greys greys unconfined_u:object_r:ssh_home_t:s0 95 Sep  6 20:28 /home/greys/.ssh/authorized_keys
```

That's the answer! We need to change **/var/ssh/greys/authorized_keys** file to the **ssh_home_t** context.

Updating SELinux context for a file

First, let's change the SELinux context:

```
root@rhel8:~ # semanage fcontext -a -t ssh_home_t
```

```
/var/ssh/greys/authorized_keys
```

… and now we relabel the actual file:

```
root@rhel8:~ # restorecon -Rv /var/ssh/greys/authorized_keys
Relabeled    /var/ssh/greys/authorized_keys    from
system_u:object_r:var_t:s0 to system_u:object_r:ssh_home_t:s0
```

That’s it – after that my logins using SSH keys started working just fine. Hope you find this example useful!

See Also

- Confirm Current SELinux Mode
- How To Disable SELinux
- How To: List SELinux Contexts for Files
- Where To Learn More About SELinux