Module 7 - Package management & troubleshooting

By Juan Medina

jmedina@collin.edu

Package Management

One of the main differences we will find among Linux Distributions that will make us decide for one over the other is the packaging system and the vitality of the distribution's community. As we spend more time with Linux, we see that its software landscape is extremely dynamic. Things are constantly changing.

Package management is a method of installing and maintaining software on the system. In the early days of Linux one had to download and compile source code in order to install a software. Having access to source code is the great wonder of Linux. It gives us the ability to examine and improve the system. But having a pre-compiled ready to install package it's faster and much easier to manage.

While all of the major distributions provide powerful and sophisticated graphical programs for maintaining the system, it is important to learn about the command-line programs, too. They can perform many tasks that are impossible to do with their graphical counterparts.

Different distributions use different packaging systems and in most cases a package intended for one distribution is not compatible with another distribution. Most distributions fall into one of two packaging technologies: the Debian ".deb" or the Red Hat ".rpm". There are some important exceptions such as Gentoo, Slackware, and Foresight, but most others use one of these two as base system.

- **Debian Style (.deb)** Debian, Ubuntu, Xandros, Linspire
- **Red Hat Style (.rpm)** Fedora, CentOS, Red Hat Enterprise Linux, OpenSUSE, Mandriva, PCLinuxOS

How A Package System Works

The windows method of software distribution usually entails buying a piece of installation media and then running an **"installation wizard"**. Linux doesn't work that way. Virtually all software for a Linux system will be found on the Internet. Most of it will be provided by the distribution vendor in the form of **package files** and the rest will be available in **source code** form that can be installed manually or **specialized packages** that either run as is or have their own installation methodology.

Package Files

A package file is a compressed collection of files that comprise the software package. It may consist of numerous programs and data files that support the programs. In addition to the files to be installed, the package file also includes metadata about the package, such as a text description of the package and its contents. Additionally, many packages contain pre- and post-installation scripts that perform configuration tasks before and after the package installation.

Repositories

Packages are made available to the users in central repositories that may contain many thousands of packages, each specially built and maintained for the distribution. There might be multiple versions of a package and might also have third-party packages.

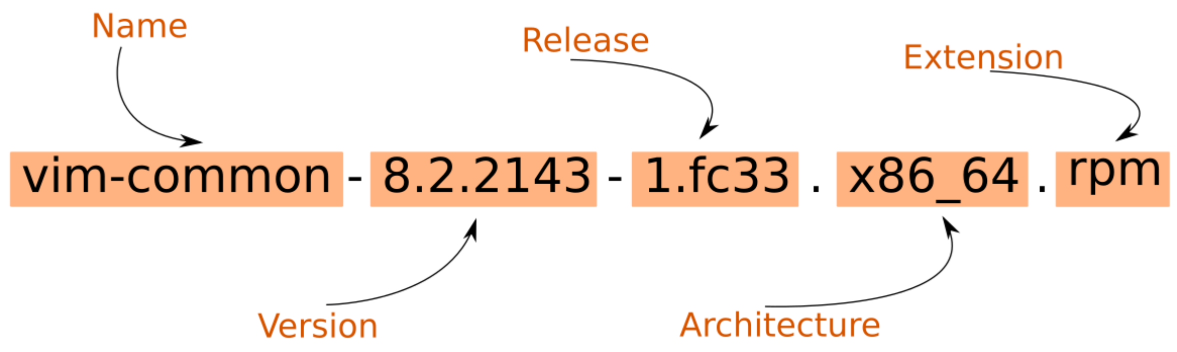
Dependencies

Remember that everything in Linux must be small, and there is no need to solve something that was already solved by somebody else!. Common activities, such as input/output, for example, are handled by **shared routines** by many programs. These routines are stored in what it's known as **shared libraries**, which provide essential services to more than one program. If a package requires a shared resource such as a shared library, it is said to have a **dependency**. Modern package management systems all provide some method of dependency resolution to ensure that when a package is installed, all of its dependencies are installed, too.

RPMs

The RPM Package Manager, developed by Red Hat, provides a standard way to package software for distribution. Managing software in the form of RPM packages is much simpler than compiling from code. RPM stores Information about packages in a local database.

RPM package files names consist of five elements:



Only the package name is required for installing packages from repositories. If multiple versions exist, the package with the higher version number is installed. If multiple releases of a single version exist, the package with the higher release number is installed.

Each RPM package is made up of three components:

- The files that will be installed.
- Information about the package (metadata).
- Scripts that may run when this package is installed, updated, or removed.

Typically, software providers digitally sign RPM packages using GPG keys. The RPM system accepts or rejects by verifiying the package integrity by confirming that the package was signed by the appropriate GPG key.

Examining RPM packages

The rpm utility is a low-level tool that can get information about the contents of package files and installed packages.

The general form of a query is:

- **rpm -q [select-options] [query-options]**

List all installed packages

```
[jmedinar@localhost ~]$ rpm -qa
pulseaudio-module-bluetooth-14.0-2.fc33.x86_64
libbluray-1.2.1-2.fc33.x86_64
tesseract-tessdata-doc-4.0.0-9.fc33.noarch
lohit-assamese-fonts-2.91.5-10.fc33.noarch
perl-XML-Writer-0.625-18.fc33.noarch
gnome-calculator-3.38.2-1.fc33.x86_64
libwpg-0.3.3-5.fc33.x86_64
libreoffice-help-en-7.0.4.2-1.fc33.x86_64
```

Find out what package provides FILENAME

```
[jmedinar@localhost ~]$ rpm -qf /etc/yum.repos.d
fedora-repos-33-1.noarch
```

List what version of the package is currently installed

```
[jmedinar@localhost ~]$ rpm -q dnf
dnf-4.4.2-1.fc33.noarch
```

Get detailed information about the package

```
[jmedinar@localhost ~]$ rpm -qi dnf
Name         : dnf
Version      : 4.4.2
Release      : 1.fc33
Architecture: noarch
Install Date: Sat 19 Dec 2020 01:08:18 PM CST
Group        : Unspecified
Size         : 2076833
License      : GPLv2+
Signature    : RSA/SHA256, Fri 27 Nov 2020 09:42:35 AM CST, Key ID
49fd77499570ff31
Source RPM   : dnf-4.4.2-1.fc33.src.rpm
Build Date   : Thu 26 Nov 2020 04:17:03 PM CST
Build Host   : buildvm-x86-30.iad2.fedoraproject.org
Packager     : Fedora Project
Vendor       : Fedora Project
URL          : https://github.com/rpm-software-management/dnf
Bug URL      : https://bugz.fedoraproject.org/dnf
Summary      : Package manager
Description :
Utility that allows users to manage packages on their systems.
It supports RPMs, modules and comps groups & environments.
```

List the files installed by the package

```
[jmedinar@localhost ~]$ rpm -ql dnf
/usr/bin/dnf
/usr/lib/systemd/system/dnf-makecache.service
/usr/lib/systemd/system/dnf-makecache.timer
/usr/share/bash-completion
/usr/share/bash-completion/completions
/usr/share/bash-completion/completions/dnf
/usr/share/locale/ar/LC_MESSAGES/dnf.mo
/usr/share/locale/bg/LC_MESSAGES/dnf.mo
/usr/share/locale/bn_IN/LC_MESSAGES/dnf.mo
/usr/share/locale/ca/LC_MESSAGES/dnf.mo
... output ommitted ...
```

List just the configuration files installed by the package

```
[jmedinar@localhost ~]$ rpm -qc openssh-server
/etc/pam.d/sshd
/etc/ssh/sshd_config
/etc/ssh/sshd_config.d/50-redhat.conf
/etc/sysconfig/sshd
```

List just the documentation files installed by the package

```
[jmedinar@localhost ~]$ rpm -qd openssh-clients
/usr/share/man/man1/scp.1.gz
/usr/share/man/man1/sftp.1.gz
/usr/share/man/man1/ssh-add.1.gz
/usr/share/man/man1/ssh-agent.1.gz
/usr/share/man/man1/ssh-copy-id.1.gz
/usr/share/man/man1/ssh-keyscan.1.gz
/usr/share/man/man1/ssh.1.gz
/usr/share/man/man5/ssh_config.5.gz
/usr/share/man/man8/ssh-pkcs11-helper.8.gz
/usr/share/man/man8/ssh-sk-helper.8.gz
```

Show the shell scripts that run before or after the package is installed or removed

```
[jmedinar@localhost ~]$ rpm -q --scripts openssh-server

preinstall scriptlet (using /bin/sh):
getent group sshd >/dev/null || groupadd -g 74 -r sshd || :
getent passwd sshd >/dev/null || \
  useradd -c "Privilege-separated SSH" -u 74 -g sshd \
  -s /sbin/nologin -r -d /var/empty/sshd sshd 2> /dev/null || :
postinstall scriptlet (using /bin/sh):


if [ $1 -eq 1 ] && [ -x /usr/bin/systemctl ]; then
        # Initial installation
        /usr/bin/systemctl --no-reload preset sshd.service sshd.socket || :
fi

# Migration scriptlet for Fedora 31 and 32 installations to sshd_config
# drop-in directory (in F32+).
# Do this only if the file generated by anaconda exists, contains our config
# directive and sshd_config contains include directive as shipped in our package
test -f /etc/sysconfig/sshd-permitrootlogin && \
  test ! -f /etc/ssh/sshd_config.d/01-permitrootlogin.conf && \
  grep -q '^PERMITROOTLOGIN="-oPermitRootLogin=yes"' /etc/sysconfig/sshd-permitrootlogin && \
  grep -q '^Include /etc/ssh/sshd_config.d/\*.conf' /etc/ssh/sshd_config && \
  echo "PermitRootLogin yes" >> /etc/ssh/sshd_config.d/25-permitrootlogin.conf && \
  rm /etc/sysconfig/sshd-permitrootlogin || :
preuninstall scriptlet (using /bin/sh):


if [ $1 -eq 0 ] && [ -x /usr/bin/systemctl ]; then
        # Package removal, not upgrade
        /usr/bin/systemctl --no-reload disable --now sshd.service sshd.socket || :
```

```
fi
postuninstall scriptlet (using /bin/sh):


if [ $1 -ge 1 ] && [ -x /usr/bin/systemctl ]; then
        # Package upgrade, not uninstall
        /usr/bin/systemctl try-restart sshd.service || :
fi
```

list change information for the package

```
[jmedinar@localhost ~]$ rpm -q --changelog vim-common | head

* Mon Dec 14 2020 Zdenek Dohnal <zdohnal@redhat.com> - 2:8.2.2143-1
- patchlevel 2143

* Mon Dec 14 2020 Zdenek Dohnal <zdohnal@redhat.com> - 2:8.2.2115-2
- 1907335 - installing vim no longer works, due to package conflicts with vim-minimal

* Wed Dec 09 2020 Zdenek Dohnal <zdohnal@redhat.com> - 2:8.2.2115-1
- patchlevel 2115

* Wed Dec 09 2020 Zdenek Dohnal <zdohnal@redhat.com> - 2:8.2.2108-2
... output ommitted ...
```

By default, RPM gets information from the local database of installed packages. However, you can use the -p option to specify that you want to get information about a downloaded package file. You might want to do this in order to inspect the contents of the package file before installing it.

```
[jmedinar@localhost Downloads]$ ls -l *rpm
-rw-r--r--. 1 jmedinar jmedinar 51812 Dec  2 18:30 python3-nmap-0.6.1-9.fc29.noarch.rpm

[jmedinar@localhost Downloads]$ rpm -qpif python3-nmap-0.6.1-9.fc29.noarch.rpm
warning: Found bdb Packages database while attempting sqlite backend: using bdb backend.
warning: python3-nmap-0.6.1-9.fc29.noarch.rpm: Header V3 RSA/SHA256 Signature, key ID 429476b4: NOKEY
Name        : python3-nmap
Version     : 0.6.1
Release     : 9.fc29
Architecture: noarch
Install Date: (not installed)
Group       : Unspecified
Size        : 190902
License     : GPLv3+
Signature   : RSA/SHA256, Wed 18 Jul 2018 08:57:40 AM CDT, Key ID a20aa56b429476b4
Source RPM  : python-nmap-0.6.1-9.fc29.src.rpm
Build Date  : Sat 14 Jul 2018 11:44:18 PM CDT
Build Host  : buildvm-13.phx2.fedoraproject.org
```

```
Packager    : Fedora Project
Vendor      : Fedora Project
URL         : http://xael.org/norman/python/python-nmap/
Bug URL     : https://bugz.fedoraproject.org/python-nmap
Summary     : A python library which helps in using nmap port
scanner
Description :
python3-nmap is a python library which helps in using nmap port
scanner. It
allows to easily manipulate nmap scan results and will be a perfect
tool
for systems administrators who want to automatize scanning task and
reports.
It also supports nmap script outputs.
```

Installing RPM packages

The rpm command can also be used to install an RPM package that you have downloaded to your local directory.

```
[root@localhost ~]# rpm -ivh wonderwidgets-1.0-4.x86_64.rpm
Verifying...                ################################# [100%]
Preparing...                ################################# [100%]
Updating / installing...
   1:wonderwidgets-1.0-4    ################################# [100%]
```

Updating Software with RPM Packages

RPM packages are always a complete software. Linux does not require that older packages be installed and then patched. To update software, RPM removes the older version of the package and installs the new version. Updates usually retain configuration files, but the packager of the new version defines the exact behavior.

DNF

The **DNF command** (Dandified yum) It is the default package manager for Red Hat based systems. It does Package Management using RPM and libsolv. If you where already familiar with the `Yum` tool then you need to know that it has been deprecated and replaced by DNF. It brings some significant changes:

- Faster, more mathematically correct method for solving dependency resolution
- A "clean", well documented Python API with C bindings &
- Python 3 support

The shift solves old dependency solving problems and leaves DNF ready for future challenges in the devops world.

Installing packages

`dnf install PACKAGENAME` obtains and installs a software package, including any dependencies.

```
[jmedinar@localhost ~]$ sudo dnf install mariadb
Fedora 33 - x86_64 - VirtualBox                                           10 kB/s | 6.9 kB     00:00
Last metadata expiration check: 0:43:11 ago on Mon 28 Dec 2020 09:10:11 AM CST.
Dependencies resolved.
================================================================================================================
 Package                  Architecture            Version                   Repository                Size
================================================================================================================
Installing:
 mariadb                        x86_64    3:10.4.17-1.fc33                     updates                1.8 M
Installing dependencies:
 mariadb-common                 x86_64    3:10.4.17-1.fc33                     updates                 33 k

Transaction Summary
================================================================================================================
Install  2 Packages

Total download size: 1.9 M
Installed size: 38 M
Is this ok [y/N]: y
Downloading Packages:
(1/2): mariadb-common-10.4.17-1.fc33.x86_64.rpm         83 kB/s |  33 kB     00:00
(2/2): mariadb-10.4.17-1.fc33.x86_64.rpm               2.6 MB/s | 1.8 MB     00:00
----------------------------------------------------------------------------------------------------------------
Total                                                   1.8 MB/s | 1.9 MB     00:01

Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Preparing        : 1/1
  Installing       : mariadb-common-3:10.4.17-1.fc33.x86_64                                               1/2
  Installing       : mariadb-3:10.4.17-1.fc33.x86_64                                                       2/2
  Running scriptlet: mariadb-3:10.4.17-1.fc33.x86_64                                                       2/2
  Verifying        : mariadb-3:10.4.17-1.fc33.x86_64
 1/2
  Verifying        : mariadb-common-3:10.4.17-1.fc33.x86_64
2/2

Installed:
  mariadb-3:10.4.17-1.fc33.x86_64       mariadb-common-3:10.4.17-
1.fc33.x86_64

Complete!
```

Installing groups

dnf also has the concept of groups, which are collections of related software installed together for a particular purpose. In Red Hat, there are two kinds of groups:

- Regular groups are collections of packages.
- Environment groups are collections of regular groups.

Like dnf list, the dnf group list command shows the names of installed and available groups.

```
[jmedinar@localhost ~]$ dnf group list
Available Environment Groups:
   Fedora Custom Operating System
   Minimal Install
   Fedora Server Edition
   Fedora Workstation
   Basic Desktop
...output omitted...
Installed Groups:
   Container Management
   LibreOffice
   GNOME Desktop Environment
   Fonts
   Hardware Support
Available Groups:
   3D Printing
   Administration Tools
   Audio Production
   Authoring and Publishing
   C Development Tools and Libraries
   Cloud Infrastructure
...output omitted...
```

Some groups are normally installed through environment groups and are hidden by default. List these hidden groups with the yum group list hidden command.

```
[jmedinar@localhost ~]$ dnf group list hidden
Available Environment Groups:
   Fedora Custom Operating System
   Minimal Install
   Fedora Server Edition
   Fedora Workstation
   Basic Desktop
...output omitted...
Installed Groups:
   Container Management
   LibreOffice
   GNOME Desktop Environment
   Fonts
   Hardware Support
Available Groups:
   3D Printing
   Administration Tools
   Audio Production
   Authoring and Publishing
   C Development Tools and Libraries
   Cloud Infrastructure
...output omitted...
```

`dnf group info` displays information about a group. It includes a list of mandatory, default, and optional package names.

```
[jmedinar@localhost ~]$ dnf group info "RPM Development Tools"
Group: RPM Development Tools
 Description: These tools include core development tools such
rpmbuild.
 Mandatory Packages:
   redhat-rpm-config
   rpm-build
 Default Packages:
   koji
   mock
   rpmdevtools
 Optional Packages:
   pungi
   rpmlint
```

- yum group install installs a group that installs its mandatory and default packages and the packages they depend on

```
[jmedinar@localhost ~]$ sudo dnf group install "RPM Development Tools"
...output omitted...
Installing Groups:
   RPM Development Tools
   Transaction Summary
===========================================================================
   Install 64 Packages
   Total download size: 21 M
   Installed size: 62 M
   Is this ok [y/N]: y
...output omitted...
```

Searching Packages

Using the high-level tools to search repository meta-data, a package can be located based on its name or description.

dnf list displays installed and available packages.

```
[jmedinar@localhost ~]$ dnf list 'http*' | head
Installed Packages
httpd.x86_64                                2.4.46-1.fc33
     @fedora
httpd-filesystem.noarch                     2.4.46-1.fc33
     @fedora
httpd-tools.x86_64                          2.4.46-1.fc33
     @fedora
Available Packages
http-parser.i686                            2.9.4-3.fc33
      fedora
http-parser.x86_64                          2.9.4-3.fc33
      fedora
http-parser-devel.i686                      2.9.4-3.fc33
      fedora
http-parser-devel.x86_64                    2.9.4-3.fc33
      fedora
```

`dnf search KEYWORD` lists packages by keywords found in the name and summary fields only

```
[jmedinar@localhost ~]$ yum search all "web server"
================================= Summary & Description Matched:
web server =================================
aws.i686 : Ada Web Server
aws.x86_64 : Ada Web Server
caddy.x86_64 : Web server with automatic HTTPS
compat-golang-github-mholt-caddy-1-devel.noarch : HTTP/2 web server
with automatic HTTPS
erlang-inets.x86_64 : A set of services such as a Web server and a
HTTP client etc
```

```
ghc-snap-server.x86_64 : A web server for the Snap Framework
...output omitted...
================================= Summary Matched: web server
=================================
R-presser.x86_64 : Lightweight Web Server for Testing
ghc-warp.x86_64 : A fast, light-weight web server for WAI
applications
libcurl.x86_64 : A library for getting files from web servers
libcurl.i686 : A library for getting files from web servers
logstalgia.x86_64 : Web server access log visualizer
...output omitted...
================================= Description Matched: web server
=================================
R-RCurl.x86_64 : General network (HTTP/FTP) client interface for R
acme-tiny.noarch : Tiny auditable script to issue, renew Let's
Encrypt certificates
awstats.noarch : Advanced Web Statistics
cronolog.x86_64 : Web log rotation program for Apache
elog.x86_64 : Logbook system to manage notes through a Web
interface
fail2ban.noarch : Daemon to ban hosts that cause multiple
authentication errors
freeradius.x86_64 : High-performance and highly configurable free
RADIUS server
...output omitted...
```

or a single package

```
[jmedinar@localhost ~]$ yum search httpd
================================= Name Exactly Matched: httpd
=================================
httpd.x86_64 : Apache HTTP Server
...output omitted...
```

`dnf info PACKAGENAME` returns detailed information about a package, including the disk space needed for installation.

To get information on the Apache HTTP Server:

```
[jmedinar@localhost ~]$ dnf info httpd
Installed Packages
Name         : httpd
Version      : 2.4.46
Release      : 1.fc33
Architecture : x86_64
Size         : 4.6 M
Source       : httpd-2.4.46-1.fc33.src.rpm
Repository   : @System
From repo    : fedora
Summary      : Apache HTTP Server
URL          : https://httpd.apache.org/
License      : ASL 2.0
Description  : The Apache HTTP Server is a powerful, efficient, and
extensible web server.
```

`dnf provides PATHNAME` displays packages that match the path name specified.

To find packages that provide the /var/www/html directory, use:

```
[jmedinar@localhost ~]$ dnf provides /var/www/html
httpd-filesystem-2.4.46-1.fc33.noarch : The basic directory layout for the Apache HTTP Server
Repo        : @System
Matched from:
Filename    : /var/www/html

httpd-filesystem-2.4.46-1.fc33.noarch : The basic directory layout for the Apache HTTP Server
Repo        : fedora
Matched from:
Filename    : /var/www/html
```

Enabling software repositories

To view only enabled repositories:

```
[jmedinar@localhost ~]$ dnf repolist
repo id                                 repo name
docker-ce-stable                        Docker CE Stable - x86_64
fedora                                  Fedora 33 - x86_64
fedora-cisco-openh264                   Fedora 33 openh264 (From
Cisco) - x86_64
fedora-modular                          Fedora Modular 33 - x86_64
google-chrome                           google-chrome
rpmfusion-free                          RPM Fusion for Fedora 33 -
Free
rpmfusion-free-updates                  RPM Fusion for Fedora 33 -
Free - Updates
rpmfusion-nonfree                       RPM Fusion for Fedora 33 -
Nonfree
rpmfusion-nonfree-updates               RPM Fusion for Fedora 33 -
Nonfree - Updates
slack                                   slack
teams                                   teams
updates                                 Fedora 33 - x86_64 -
Updates
updates-modular                         Fedora Modular 33 - x86_64
- Updates
```

To view all available repositories add the `all` option

```
[jmedinar@localhost ~]$ dnf repolist all
repo id                   repo name                           status
docker-ce-stable          Docker CE Stable - x86_64
disabled
fedora                    Fedora 33 - x86_64                  enabled
fedora-source             Fedora 33 - Source
disabled
google-chrome             google-chrome                       enabled
rpmfusion-free            RPM Fusion for Fedora 33 - Free     enabled
...output omitted...
```

The `dnf config-manager` command can be used to enable or disable repositories. To enable a repository, the command sets the enabled parameter to 1. For example, the following command enables the `docker-ce-stable` repository:

```
[root@localhost ~]# dnf config-manager --enable docker-ce-stable
[root@localhost ~]# dnf repolist | grep docker
docker-ce-stable                Docker CE Stable - x86_64
```

To enable support for a new third-party repository, create a file in the `/etc/yum.repos.d/` directory. Repository configuration files must end with a `.repo` extension. The repository definition contains the URL of the repository, a name, whether to use GPG to check the package signatures, and if so, the URL pointing to the trusted GPG key.

Creating Repositories

Create repositories with the `dnf config-manager` command.

The following command creates a file named `/etc/yum.repos.d/dl.fedoraproject.org_pub_epel_8_x86_64_.repo` with the output shown.

```
[root@localhost ~]# ls -l /etc/yum.repos.d/ | grep fedoraproject
[root@localhost ~]# dnf config-manager --add-
repo="http://dl.fedoraproject.org/pub/epel/8/x86_64/"
Adding repo from: http://dl.fedoraproject.org/pub/epel/8/x86_64/
[root@localhost ~]# ls -l /etc/yum.repos.d/ | grep fedoraproject
-rw-r--r--. 1 root root  194 Dec 28 11:30
dl.fedoraproject.org_pub_epel_8_x86_64_.repo
```

We have created the following repository

```
[dl.fedoraproject.org_pub_epel_8_x86_64_]
name=created by dnf config-manager from
http://dl.fedoraproject.org/pub/epel/8/x86_64/
baseurl=http://dl.fedoraproject.org/pub/epel/8/x86_64/
enabled=1
```

We can now enable it or disable it

```
[root@localhost ~]# dnf repolist
repo id                                    repo name
dl.fedoraproject.org_pub_epel_8_x86_64_   created by dnf config-
manager http://dl.fedoraproject.org/pub/epel/8...
docker-ce-stable                           Docker CE Stable - x86_64
```

Updating Packages

Making sure that all of the installed software on a machine stays up to date would be an enormous undertaking without a package system. You would have to track upstream changes and security alerts for hundreds of different packages. While a package manager doesn't solve every problem you'll encounter when upgrading software, it does enable you to maintain most system components with just a few commands.

`dnf update PACKAGENAME` obtains and installs a newer version of the specified package, including any dependencies. Generally the process tries to preserve configuration files in place, but in some cases, they may be renamed if the packager thinks the old one will not work after the update. With no PACKAGENAME specified, it installs all relevant updates.

```
[root@localhost]# dnf update
Dependencies resolved.
Nothing to do.
Complete!
```

Since a new kernel can only be tested by booting into that kernel, the package is specifically designed so that multiple versions may be installed at once. If the new kernel fails to boot, the old kernel is still available. Using yum update kernel will actually install the new kernel. The configuration files hold a list of packages to always install even if the administrator requests an update.

To list the installed kernels

```
[root@localhost]# dnf list kernel
Installed Packages
kernel.x86_64                5.9.14-100.fc32                @updates
kernel.x86_64                5.9.14-200.fc33                @updates
kernel.x86_64                5.9.16-200.fc33                @updates
```

To view the current running kernel

```
[root@localhost]# uname -r
5.9.16-200.fc33.x86_64
```

To update the kernel

```
[root@localhost ~]# dnf update kernel
```

Uninstalling

Since a package manager knows what files are provided by a given package, it can usually remove them cleanly from a system if the software is no longer needed.

`dnf remove PACKAGENAME` removes an installed software package, including any supported packages.

```
[root@localhost ~]# dnf remove httpd
Dependencies resolved.
=========================================
 Package                 Architecture            Version
    Repository               Size
=========================================
Removing:
 httpd                   x86_64                  2.4.46-1.fc33
    @fedora                  4.6 M
Removing dependent packages:
 gnome-user-share        x86_64                  3.34.0-4.fc33
    @fedora                  307 k
Removing unused dependencies:
 fedora-logos-httpd      noarch                  30.0.2-5.fc33
    @fedora                  9.9 k
 httpd-filesystem        noarch                  2.4.46-1.fc33
    @fedora                  400
 httpd-tools             x86_64                  2.4.46-1.fc33
    @fedora                  201 k
 mod_dnssd               x86_64                  0.6-23.fc33
    @fedora                  57 k
 mod_http2               x86_64                  1.15.14-2.fc33
   @fedora                  372 k

Transaction Summary
=========================================
Remove  7 Packages

Freed space: 5.5 M
Is this ok [y/N]:
```

Installing software from Source Code

From time to time, you may want or need to compile a software from the source code. This involves compiling C or C++ code into something we can execute in your Linux specific architecture.

There are some advantages of installing software using the Source code:

- You can use the source codes to modify the package.
- Research and understand the coding algorithm
- You can compile the code specifically for your architecture
    - It will be faster
    - More efficient

Find the software

Downloading the source code from a website or cloning a git repository.

I am taking as an example `nmap` from the website https://nmap.org/

- Go to Download
- Scroll down to the `Source Code Distribution` section
- Search for the **Latest stable Nmap release tarbal** and download the package

This is very specific of every website so in general you will have to find the package.

Acquiring the package from GitHub is a little bit more straight forward because the steps are always the same you just have to find the proper repo.

I am taking as an example `wxMEdit` from https://github.com/wxMEdit/wxMEdit

- Click on the `Code` green button
- Select HTTPS mode
- Copy the url provided
- In your Terminal window create a folder an run the following command

```
[root@localhost ~]# mkdir test
[root@localhost ~]# cd test
[root@localhost test]# git clone
https://github.com/wxMEdit/wxMEdit.git
Cloning into 'wxMEdit'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 12879 (delta 1), reused 4 (delta 1), pack-reused
12873
Receiving objects: 100% (12879/12879), 16.02 MiB | 9.27 MiB/s,
done.
Resolving deltas: 100% (9569/9569), done.
[root@localhost test]# ls
wxMEdit
```

If git is not installed in your computer you can install it with the following command

```
[root@localhost]# dnf install git
```

Installing the software

The typical method of operation is the execution of these three commands:

```
[jmedinar@localhost ~]$ ./configure
[jmedinar@localhost ~]$ make
[jmedinar@localhost ~]$ sudo make install
```

The first command runs the configure script that analyses the libraries installed on your system. If a required library isn't installed, it will report it and you will need to install it before you can continue (dependencies, remember?). Not only will you need the library installed, but also the development files must be present as well. Source code files use the functionality found in these libraries.

After the configure command is executed, a Makefile will be created. By running 'make', it will read the Makefile in the current directory and start running the compiler (gcc for C, or g++ for C++) to compile the software.

The third command isn't strictly essential but is recommended for system-wide access, that is running the executable from anywhere in the terminal. This command tells 'make' to run instructions for installing the program to the whole system.

Graphical Installers

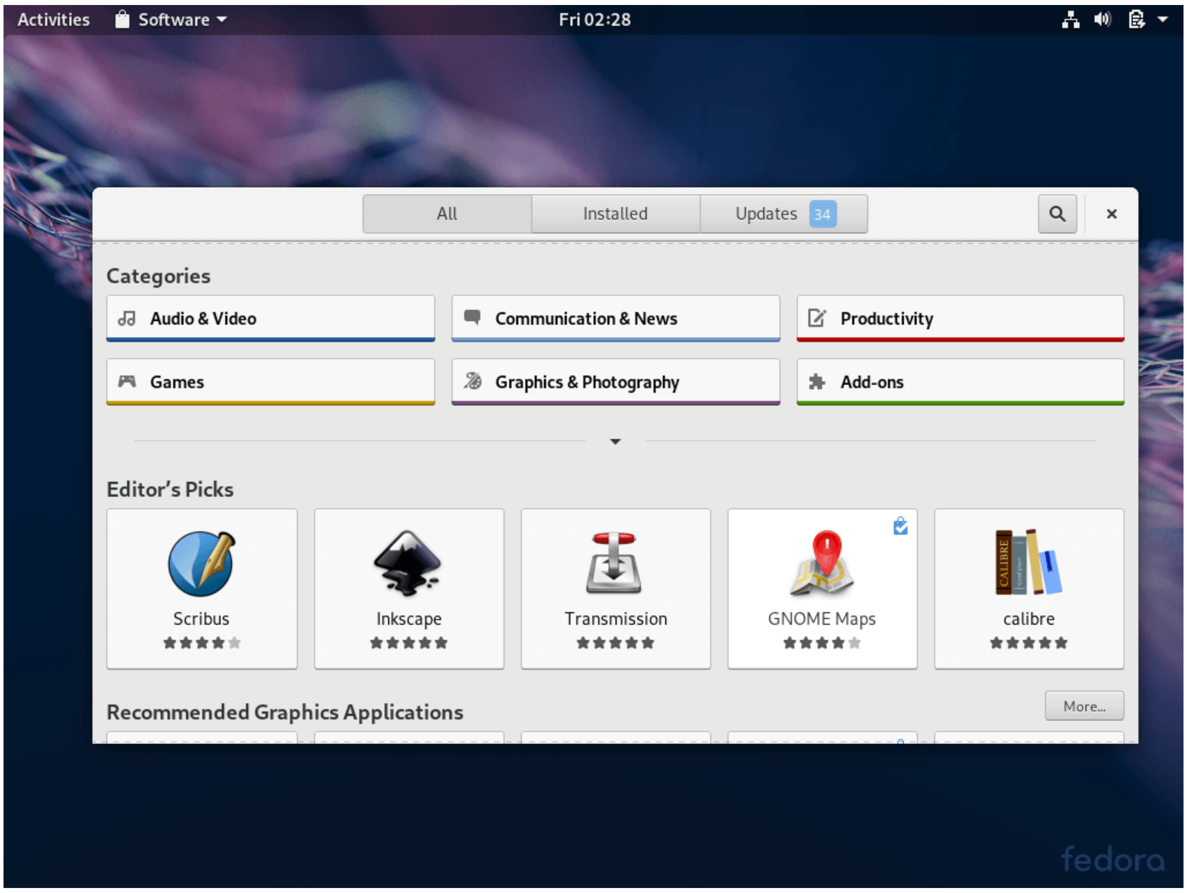
You can use a graphical utility to browse the available software packages. When you find the software that you want, you can use the utility to install it on your Fedora system.

Before you start

To install any packages on your Fedora system, you need to have **root** privileges.

Procedure

1. On your GNOME desktop, select the **Activities** menu and then click the icon.
2. Find a software package in one of the following ways:
   - Click one of the suggested categories, for example, **Productivity**. Then review the suggested packages. To see more packages, select items in the menu in the left-hand part of the window. For example, for the Productivity category, you can select **Calendar**, **Database**, **Finance**, **Word Processor**.
   - Choose one of the **Editor's Picks** or other recommended software in the window.
   - Click the icon, then enter a keyword or the name of the application. Review the suggested packages.
3. Click a package to read its description.
4. To install the package, click the **Install** button. When prompted, provide the root password.


Activities
Software
Fri 02:28
All
Installed
Updates 34
Categories
Audio & Video
Communication & News
Productivity
Games
Graphics & Photography
Add-ons
Editor's Picks
Scribus
Inkscape
Transmission
GNOME Maps
calibre
Recommended Graphics Applications
More...
fedora

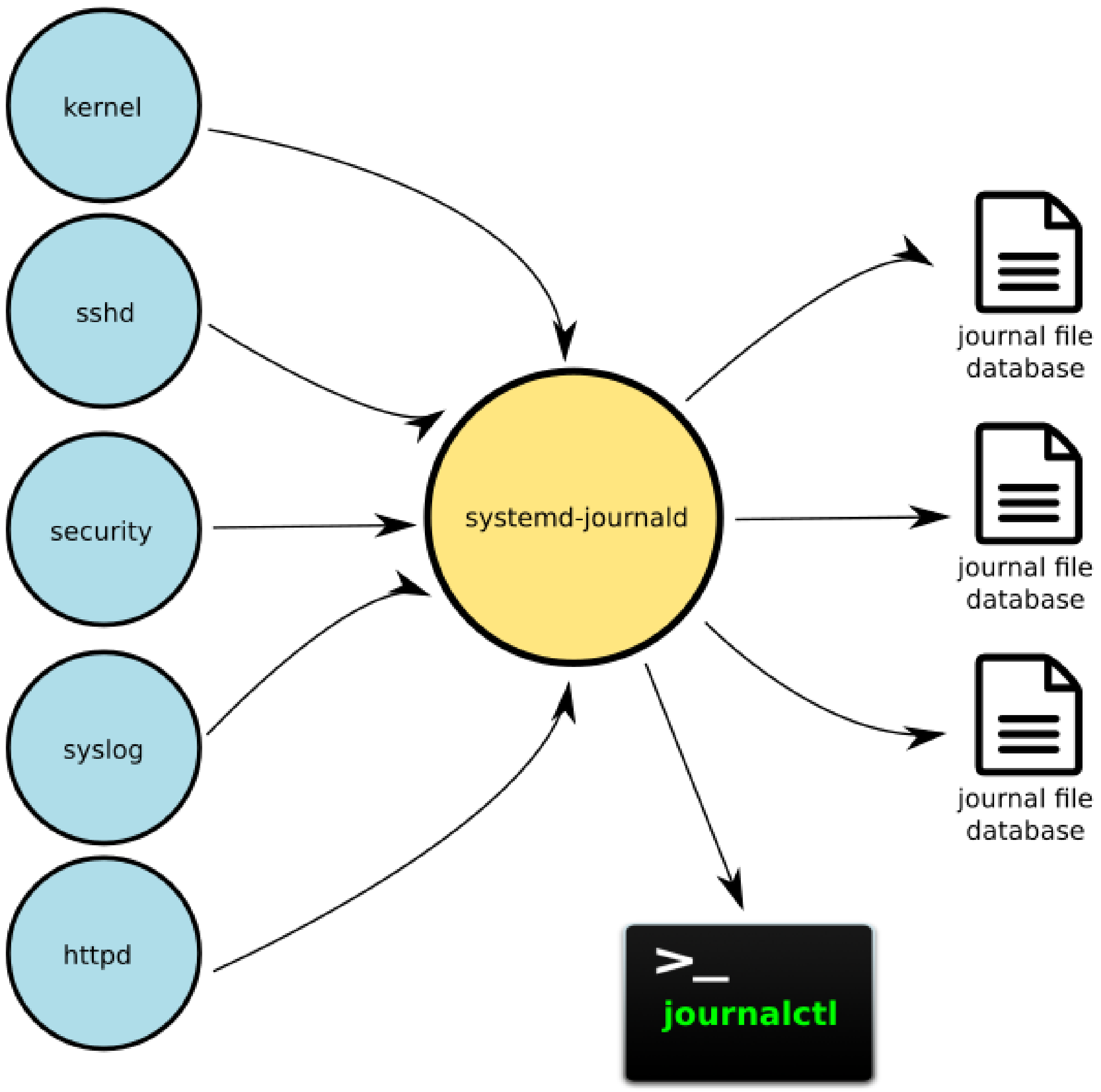
System Logs

When troubleshooting Linux there are multiple things to check besides performance, hardware, and software we can also take a look at the messages that the systems are sending to us constantly by looking into their logs!. These logs are used to audit the system and troubleshoot problems. Many systems record logs of events in text files which are kept in the `/var/log` directory. These logs can be inspected using normal text utilities such as `less` and `tail`.

LOG FILE	TYPE OF MESSAGES STORED
/var/log/messages	Most syslog messages are logged here. Exceptions include messages related to authentication and email processing, scheduled job execution, and those which are purely debugging-related.
/var/log/secure	Syslog messages related to security and authentication events.
/var/log/maillog	Syslog messages related to the mail server.
/var/log/cron	Syslog messages related to scheduled job execution.
/var/log/boot.log	Non-syslog console messages related to system startup.

```
[jmedinar@localhost ~]$ sudo tail /var/log/boot.log
         Starting Virtualization daemon...
         Starting Notify NFS peers of a restart...
         Starting Permit User Sessions...
[  OK  ] Started Notify NFS peers of a restart.
[  OK  ] Finished Permit User Sessions.
[  OK  ] Started Deferred execution scheduler.
[  OK  ] Started Command Scheduler.
         Starting GNOME Display Manager...
         Starting Hold until boot process finishes up...
[  OK  ] Started GNOME Display Manager.
```

A standard logging system based on the `syslog` protocol is built into Linux. Many programs use this system to record events and organize them into log files. The `systemd-journald` and `rsyslog` services handle the `syslog` messages in Linux.



The `systemd-journald` service is at the heart of the operating system event logging architecture. It collects event messages from many sources including the kernel, output from the early stages of the boot process, standard output and standard error from daemons as they start up and run, and `syslog` events. It then restructures them into a standard format, and writes them into a structured, indexed system journal.

syslog

Many programs use the `syslog` protocol to log events to the system. Each log message is categorized by a facility and a priority.

The following table lists the standard eight `syslog` priorities from highest to lowest.

CODE	PRIORITY	SEVERITY
0	emerg	System is unusable
1	alert	Action must be taken immediately
2	crit	Critical condition
3	err	Non-critical error condition
4	warning	Warning condition
5	notice	Normal but significant event
6	info	Informational event
7	debug	Debugging-level message

The `rsyslog` service uses the facility and priority of log messages to determine how to handle them. This is configured by rules in the `/etc/rsyslog.conf` file and any file in the `/etc/rsyslog.d` directory that has a file name extension of `.conf`.

Each rule that controls how to sort `syslog` messages is a line in one of the configuration files. The left side of each line indicates the facility and severity of the `syslog` messages the rule matches. The right side of each line indicates what file to save the log message in. An asterisk (*) is a wild-card that matches all values.

For example, the following line would record messages sent to the `authpriv` facility at any priority to the file `/var/log/secure`:

```
authpriv.*                /var/log/secure
```

Analyzing the logs

Log messages start with the oldest message on top and the newest message at the end of the log file. The `rsyslog` service uses a standard format while recording entries in log files. The following example explains the anatomy of a log message in the `/var/log/secure` log file.

```
Feb 11 20:11:48 localhost sshd[1433]: Failed password for student
from 172.25.0.10 port 59344 ssh2
```

- The time stamp when the log entry was recorded
- The host from which the log message was sent
- The program or process name and `PID` number that sent the log message
- The actual message sent

Monitoring logs

Monitoring one or more log files for events is helpful to reproduce problems and issues. The tail -f /path/to/file command outputs the last 10 lines of the file specified and continues to output new lines in the file as they get written.

```
[root@localhost ~]# tail -f /var/log/secure
```

You can read regular logs using tools like vim, more, less, cat, nano, head, and tail.

The easiest and most effective is **tail,** which lets you look over log files. It’s so useful because it just displays the last bit of the logs. Which is often where you’ll find the source of the difficulty. Use `tail /var/log/messages` or `tail -f /var/log/messages`.

Sending syslog messages manually

The `logger` command can send messages to the `rsyslog` service. By default, it sends the message to the user facility with the notice priority “user.notice” unless specified otherwise with the `-p option`. It is useful to test any change to the `rsyslog` service configuration.

To send a message to the rsyslog service that gets recorded in the /var/log/boot.log log file, execute the following logger command:

```
[root@localhost ~]# logger -p local7.notice "Log entry created on host"
```

Journalctl

Some of the most compelling advantages of `systemd` are those related to process management and system logging. When using other tools, logs are usually dispersed throughout the system, handled by different daemons and processes, and can be fairly difficult to interpret when they span multiple applications. `systemd` attempts to address these issues by providing a centralized management solution for logging all kernel, applications and user processes.

The `journald` daemon collects data from all available sources and stores them in a binary format for easy and dynamic manipulation.

This gives us a number of significant advantages.

- Administrators are able to dynamically display log data according to their needs
- View data from different moments in time
- Combining the log entries sequentially from two related services to debug an issue.
- The data can be displayed in arbitrary output formats depending on what you need at the moment
- Can either be used with an existing `syslog` implementation, or it can replace the `syslog` functionality, depending on your needs.

- Centralize `syslog` on a server that you use to compile data from multiple servers.

Setting the local clocks and time zones

Correct synchronized system time is critical for log file analysis across multiple systems. The Network Time Protocol (NTP) is a standard way for machines to provide and obtain correct time information on the Internet. A machine may get accurate time information from public NTP services on the Internet, such as the NTP Pool Project. A high-quality hardware clock to serve accurate time to local clients is another option.

The `timedatectl` command shows an overview of the current time-related system settings, including current time, time zone, and NTP synchronization settings of the system.

```
[jmedinar@localhost ~]$ timedatectl
               Local time: Mon 2020-12-28 16:57:19 CST
           Universal time: Mon 2020-12-28 22:57:19 UTC
                 RTC time: Mon 2020-12-28 22:57:19
                Time zone: America/Chicago (CST, -0600)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
```

A database of time zones is available and can be listed with the `timedatectl list-timezones` command.

```
[jmedinar@localhost ~]$ timedatectl list-timezones | grep America
America/Adak
America/Anchorage
America/Araguaina
America/Argentina/Buenos_Aires
America/Argentina/Catamarca
America/Argentina/Cordoba
America/Argentina/Jujuy
America/Argentina/La_Rioja
... Output Omitted ...
```

Time zone names are based on the public time zone database that `IANA` maintains. Time zones are named based on continent or ocean, then typically but not always the largest city within the time zone region. For example, most of the US Mountain time zone is `America/Denver`.

Selecting the correct name can be non-intuitive in cases where localities inside the time zone have different daylight saving time rules. For example, in the USA, much of the state of Arizona (US Mountain time) does not have a daylight saving time adjustment at all and is in the time zone America/Phoenix.

The command `tzselect` is useful for identifying correct zoneinfo time zone names. It interactively prompts the user with questions about the system's location, and outputs the name of the correct time zone. It does not make any change to the time zone setting of the system.

```
[jmedinar@localhost ~]$ tzselect
Please identify a location so that time zone rules can be set
correctly.
Please select a continent, ocean, "coord", or "TZ".

1) Africa         5) Atlantic Ocean      9) Pacific Ocean
2) Americas       6) Australia          10) coord - I want to use
geographical coordinates.
3) Antarctica     7) Europe             11) TZ - I want to specify
the timezone using the Posix TZ format.
4) Asia           8) Indian Ocean
#? 2

Please select a country whose clocks agree with yours.
... Output Omitted ...
6) Barbados   13) Chile      20) Ecuador       27) Guyana   34)
Nicaragua   41) St Lucia   48) Turks & Caicos Is
7) Belize     14) Colombia   21) El Salvador   28) Haiti    35)
Panama      49) United States
#? 49

Please select one of the following timezones.
... Output Omitted ...
5) Eastern - IN (most areas)   11) Central (most areas)   17)
Central - ND (Mercer)   23) Alaska   29) Hawaii
#? 11

The following information has been given:

   United States
   Central (most areas)

Therefore TZ='America/Chicago' will be used.
Selected time is now:   Mon Dec 28 17:01:17 CST 2020.
Universal Time is now:  Mon Dec 28 23:01:17 UTC 2020.
Is the above information OK?
1) Yes
2) No
#? 1
```

root can change the system setting to update the current time zone using the `timedatectl set-timezone` command.

```
[root@localhost ~]# timedatectl set-timezone America/Chicago
[root@localhost ~]# timedatectl
               Local time: Mon 2020-12-28 17:07:27 CST
           Universal time: Mon 2020-12-28 23:07:27 UTC
                 RTC time: Mon 2020-12-28 23:07:27
                Time zone: America/Chicago (CST, -0600)
System clock synchronized: yes
              NTP service: active
          RTC in local TZ: no
```

Use the `timedatectl set-time` command to change the system's current time. The time is specified in the "YYYY-MM-DD hh:mm:ss" format, where either date or time can be omitted.

```
[root@localhost ~]# timedatectl set-time 9:00:00
```

The `timedatectl set-ntp` command enables or disables NTP synchronization for automatic time adjustment. The option requires either a true or false argument to turn it on or off.

```
[root@localhost ~]# timedatectl set-ntp true
```

Basic Log Viewing

To see the logs that the `journald` daemon has collected, use the `journalctl` command.

When used alone, every journal entry that is in the system will be displayed within a pager. The oldest entries will be up top:

```
[root@localhost ~]# journalctl
-- Logs begin at Wed 2020-11-04 16:26:05 CST, end at Mon 2020-12-28 17:10:07 CST. --
Nov 04 16:26:10 localhost.localdomain chronyd[1000]: System clock TAI offset set to 37 seconds
Nov 04 16:26:10 localhost.localdomain chronyd[1000]: System clock wrong by -5.519343 seconds, adjustment started
Nov 04 16:26:05 localhost.localdomain chronyd[1000]: System clock was stepped by -5.519343 seconds
Nov 04 16:26:06 localhost.localdomain chronyd[1000]: Selected source 212.51.144.46
Nov 04 16:26:07 localhost.localdomain chronyd[1000]: Source 188.165.11.86 replaced with 216.229.0.50
... Output Omitted ...
```

You will likely have pages and pages of data to scroll through, which can be tens or hundreds of thousands of lines long if `systemd` has been on your system for a long while. This demonstrates how much data is available in the journal database.

You may notice that all of the timestamps being displayed are local time. This is available for every log entry now that we have our local time set correctly on our system. All of the logs are displayed using this new information.

If you want to display the timestamps in UTC, you can use the `--utc` flag:

```
[root@localhost ~]# journalctl --utc
-- Logs begin at Wed 2020-11-04 22:26:05 UTC, end at Mon 2020-12-28
23:10:43 UTC. --
Nov 04 22:26:10 localhost.localdomain chronyd[1000]: System clock
TAI offset set to 37 seconds
Nov 04 22:26:10 localhost.localdomain chronyd[1000]: System clock
wrong by -5.519343 seconds, adjustment started
Nov 04 22:26:05 localhost.localdomain chronyd[1000]: System clock
was stepped by -5.519343 seconds
Nov 04 22:26:06 localhost.localdomain chronyd[1000]: Selected source
212.51.144.46
Nov 04 22:26:07 localhost.localdomain chronyd[1000]: Source
188.165.11.86 replaced with 216.229.0.50
... Output Omitted ...
```

Filtering by Time

While having access to such a large collection of data is definitely useful, such a large amount of information can be difficult or impossible to inspect and process mentally. Because of this, one of the most important features of `journalctl` is its filtering options.

Displaying Logs from the Current Boot

Use the `-b` flag to show you all of the journal entries that have been collected since the most recent reboot.

```
[root@localhost ~]# journalctl -b
-- Logs begin at Wed 2020-11-04 16:26:05 CST, end at Mon 2020-12-28
17:14:04 CST. --
Dec 28 11:01:26 localhost.localdomain kernel: microcode: updated
early to revision 0x21, date = 2019-02-13
Dec 28 11:01:26 localhost.localdomain kernel: Linux version 5.9.16-
200.fc33.x86_64 (mockbuild@bkernel01.iad2.fedoraproject.org) (gcc
(GCC) 10.2.1 (Red Hat 10.2.1-9), GNU ld version 2.35-15.fc33)
Dec 28 11:01:26 localhost.localdomain kernel: x86/fpu: Supporting
XSAVE feature 0x001: 'x87 floating point regs'
Dec 28 11:01:26 localhost.localdomain kernel: x86/fpu: Supporting
XSAVE feature 0x002: 'SSE registers'
Dec 28 11:01:26 localhost.localdomain kernel: x86/fpu: Supporting
XSAVE feature 0x004: 'AVX registers'
... Output Omitted ...
```

This will help you identify and manage information that is pertinent to your current environment.

Past Boots

Some distributions enable saving previous boot information by default, while others disable this feature. To enable persistent boot information, you can either create the directory to store the journal by typing:

```
[root@localhost ~]# mkdir -p /var/log/journal
```

Or you can edit the journal configuration file:

```
[root@localhost ~]# vim /etc/systemd/journald.conf
```

Under the `[Journal]` section, set the `Storage=` option to “persistent” to enable persistent logging: `` `/etc/systemd/journald.conf ``

```
. . .
[Journal]
Storage=persistent
```

When this is enabled on your server, `journalctl` provides some commands to help you work with boots as a unit of division. To see the boots that `journald` knows about, use the `--list-boots` option with `journalctl`:

```
[jmedinar@localhost ~]$ journalctl --list-boots
-2 caf0524a1d394ce0bdbcff75b94444fe Tue 2015-02-03 21:48:52 UTC—Tue
2015-02-03 22:17:00 UTC
-1 13883d180dc0420db0abcb5fa26d6198 Tue 2015-02-03 22:17:03 UTC—Tue
2015-02-03 22:19:08 UTC
 0 bed718b17a73415fade0e4e7f4bea609 Tue 2015-02-03 22:19:12 UTC—Tue
2015-02-03 23:01:01 UTC
```

This will display a line for each boot. The first column is the identifier for the boot that can be used to easily reference it.

For instance, to see the journal from the previous boot, use the `-1` relative pointer with the `-b` flag:

```
[jmedinar@localhost ~]$ journalctl -b -1
```

You can also use the boot ID to call back the data from a boot:

```
[jmedinar@localhost ~]$ journalctl -b
caf0524a1d394ce0bdbcff75b94444fe
```

Time Windows

If you need an specific windows of time that do not align well with system boots. You can filter by arbitrary time limits using the `--since` and `--until` options, which restrict the entries displayed to those after or before the given time, respectively.

The time values can come in a variety of formats. For absolute time values, you should use the following format:

```
YYYY-MM-DD HH:MM:SS
```

For instance, we can see all of the entries since January 10th, 2015 at 5:15 PM by typing:

```
[jmedinar@localhost ~]$ journalctl --since "2015-01-10 17:15:00"
```

If components of the above format are left off, some defaults will be applied. For instance, if the date is omitted, the current date will be assumed. If the time component is missing, "00:00:00" (midnight) will be substituted. The seconds field can be left off as well to default to "00":

```
[jmedinar@localhost ~]$ journalctl --since "2015-01-10" --until
"2015-01-11 03:00"
```

You can also use some named shortcuts.

- yesterday
- today
- tomorrow
- now

You do relative times by prepending "-" or "+" to a numbered value or using words like "ago" in a sentence construction.

To get the data from yesterday, you could type:

```
[jmedinar@localhost ~]$ journalctl --since yesterday
```

If you received reports of a service interruption starting at 9:00 AM and continuing until an hour ago, you could type:

```
[jmedinar@localhost ~]$ journalctl --since 09:00 --until "1 hour
ago"
```

As you can see, it's relatively easy to define flexible windows of time to filter the entries you wish to see.

Filtering by Message Interest

By Unit

We can use the `-u` option to filter in this way.

For instance, to see all of the logs from an Nginx unit on our system, we can type:

```
[jmedinar@localhost ~]$ journalctl -u nginx.service
```

Typically, you would probably want to filter by time as well in order to display the lines you are interested in.

```
[jmedinar@localhost ~]$ journalctl -u nginx.service --since today
```

This becomes extremely helpful when you take advantage of the journal's ability to interleave records from various units. For instance, if your Nginx process is connected to a PHP-FPM unit to process dynamic content, you can merge the entries from both in chronological order by specifying both units:

```
[jmedinar@localhost ~]$ journalctl -u nginx.service -u php-
fpm.service --since today
```

This can make it much easier to spot the interactions between different programs and debug systems instead of individual processes.

By Process, User, or Group ID

Some services spawn a variety of child processes to do work. If you have scouted out the exact PID of the process you are interested in, you can filter by that as well.

To do this we can filter by specifying the `_PID` field. For instance if the PID we're interested in is 8088, we could type:

```
[jmedinar@localhost ~]$ journalctl _PID=8088
```

At other times, you may wish to show all of the entries logged from a specific user or group. This can be done with the `_UID` or `_GID` filters. For instance, if your web server runs under the `www-data` user, you can find the user ID by typing:

```
[jmedinar@localhost ~]$ id -u www-data
33
```

Afterwards, you can use the ID that was returned to filter the journal results:

```
[jmedinar@localhost ~]$ journalctl _UID=33 --since today
```

The `systemd` journal has many fields that can be used for filtering. Some of those are passed from the process being logged and some are applied by `journald` using information it gathers from the system at the time of the log.

The leading underscore indicates that the `_PID` field is of the latter type. The journal automatically records and indexes the PID of the process that is logging for later filtering. You can find out about all of the available journal fields by typing:

```
[jmedinar@localhost ~]$ man systemd.journal-fields
```

By Component Path

We can also filter by providing a path location.

If the path leads to an executable, `journalctl` will display all of the entries that involve the executable in question. For instance, to find those entries that involve the `bash` executable, you can type:

```
[jmedinar@localhost ~]$ journalctl /usr/bin/bash
```

Usually, if a unit is available for the executable, that method is cleaner and provides better info (entries from associated child processes, etc). Sometimes, however, this is not possible.

Displaying Kernel Messages

Kernel messages, those usually found in `dmesg` output, can be retrieved from the journal as well.

To display only these messages, we can add the `-k` or `--dmesg` flags to our command:

```
[jmedinar@localhost ~]$ journalctl -k
```

By default, this will display the kernel messages from the current boot. You can specify an alternative boot using the normal boot selection flags discussed previously. For instance, to get the messages from five boots ago, you could type:

```
[jmedinar@localhost ~]$ journalctl -k -b -5
```

By Priority

One filter that system administrators often are interested in is the message priority. While it is often useful to log information at a very verbose level, when actually digesting the available information, low priority logs can be distracting and confusing.

You can use `journalctl` to display only messages of a specified priority or above by using the `-p` option. This allows you to filter out lower priority messages.

For instance, to show only entries logged at the error level or above, you can type:

```
[jmedinar@localhost ~]$ journalctl -p err -b
```

This will show you all messages marked as error, critical, alert, or emergency. The journal implements the standard `syslog` message levels. You can use either the priority name or its corresponding numeric value. In order of highest to lowest priority, these are:

- 0: emerg
- 1: alert
- 2: crit
- 3: err
- 4: warning
- 5: notice
- 6: info
- 7: debug

The above numbers or names can be used interchangeably with the `-p` option. Selecting a priority will display messages marked at the specified level and those above it.

Modifying the Journal Display

Truncate or Expand Output

We can adjust how `journalctl` displays data by telling it to shrink or expand the output.

By default, `journalctl` will show the entire entry in the pager, allowing the entries to trail off to the right of the screen. This info can be accessed by pressing the right arrow key.

If you'd rather have the output truncated, inserting an ellipsis where information has been removed, you can use the `--no-full` option:

```
[jmedinar@localhost ~]$ journalctl --no-full
. . .
Feb 04 20:54:13 journalme sshd[937]: Failed password for root from
83.234.207.60...h2
Feb 04 20:54:13 journalme sshd[937]: Connection closed by
83.234.207.60 [preauth]
Feb 04 20:54:13 journalme sshd[937]: PAM 2 more authentication
failures; logname...ot
```

You can also go in the opposite direction with this and tell `journalctl` to display all of its information, regardless of whether it includes unprintable characters. We can do this with the `-a` flag:

```
[jmedinar@localhost ~]$ journalctl -a
```

Output to Standard Out

By default, `journalctl` displays output in a pager for easier consumption. If you are planning on processing the data with text manipulation tools, however, you probably want to be able to output to standard output.

You can do this with the `--no-pager` option:

```
[jmedinar@localhost ~]$ journalctl --no-pager
```

This can be piped immediately into a processing utility or redirected into a file on disk, depending on your needs.

Output Formats

If you need to process journal entries in a more consumable format. You can do this using the `-o` option with a format specifier.

For instance, you can output the journal entries in JSON by typing:

```
[jmedinar@localhost ~]$ journalctl -b -u nginx -o json
```

```
{ "__CURSOR" :
"s=13a21661cf4948289c63075db6c25c00;i=116f1;b=81b58db8fd9046ab9f847d
db82a2fa2d;m=19f0daa;t=50e33c33587ae;x=e307daadb4858635",
"__REALTIME_TIMESTAMP" : "1422990364739502", "__MONOTONIC_TIMESTAMP"
: "27200938", "_BOOT_ID" : "81b58db8fd9046ab9f847ddb82a2fa2d",
"PRIORITY" : "6", "_UID" : "0", "_GID" : "0", "_CAP_EFFECTIVE" :
"3fffffffff", "_MACHINE_ID" : "752737531a9d1a9c1e3cb52a4ab967ee",
"_HOSTNAME" : "desktop", "SYSLOG_FACILITY" : "3", "CODE_FILE" :
"src/core/unit.c", "CODE_LINE" : "1402", "CODE_FUNCTION" :
"unit_status_log_starting_stopping_reloading", "SYSLOG_IDENTIFIER" :
"systemd", "MESSAGE_ID" : "7d4958e842da4a758f6c1cdc7b36dcc5",
"_TRANSPORT" : "journal", "_PID" : "1", "_COMM" : "systemd", "_EXE"
: "/usr/lib/systemd/systemd", "_CMDLINE" :
"/usr/lib/systemd/systemd", "_SYSTEMD_CGROUP" : "/", "UNIT" :
"nginx.service", "MESSAGE" : "Starting A high performance web server
and a reverse proxy server...", "_SOURCE_REALTIME_TIMESTAMP" :
"1422990364737973" }
. . .
```

This is useful for parsing with utilities. You could use the `json-pretty` format to get a better handle on the data structure before passing it off to the JSON consumer:

```
[jmedinar@localhost ~]$ journalctl -b -u nginx -o json-pretty
```

```
{
    "__CURSOR" :
"s=13a21661cf4948289c63075db6c25c00;i=116f1;b=81b58db8fd9046ab9f847
ddb82a2fa2d;m=19f0daa;t=50e33c33587ae;x=e307daadb4858635",
    "__REALTIME_TIMESTAMP" : "1422990364739502",
    "__MONOTONIC_TIMESTAMP" : "27200938",
    "_BOOT_ID" : "81b58db8fd9046ab9f847ddb82a2fa2d",
    "PRIORITY" : "6",
    "_UID" : "0",
    "_GID" : "0",
    "_CAP_EFFECTIVE" : "3fffffffff",
    "_MACHINE_ID" : "752737531a9d1a9c1e3cb52a4ab967ee",
    "_HOSTNAME" : "desktop",
    "SYSLOG_FACILITY" : "3",
    "CODE_FILE" : "src/core/unit.c",
    "CODE_LINE" : "1402",
    "CODE_FUNCTION" :
"unit_status_log_starting_stopping_reloading",
    "SYSLOG_IDENTIFIER" : "systemd",
    "MESSAGE_ID" : "7d4958e842da4a758f6c1cdc7b36dcc5",
    "_TRANSPORT" : "journal",
    "_PID" : "1",
    "_COMM" : "systemd",
    "_EXE" : "/usr/lib/systemd/systemd",
    "_CMDLINE" : "/usr/lib/systemd/systemd",
    "_SYSTEMD_CGROUP" : "/",
    "UNIT" : "nginx.service",
    "MESSAGE" : "Starting A high performance web server and a
reverse proxy server...",
    "_SOURCE_REALTIME_TIMESTAMP" : "1422990364737973"
}
```

The following formats can be used for display:

- **cat**: Displays only the message field itself.
- **export**: A binary format suitable for transferring or backing up.
- **json**: Standard JSON with one entry per line.
- **json-pretty**: JSON formatted for better human-readability
- **json-sse**: JSON formatted output wrapped to make add server-sent event compatible
- **short**: The default `syslog` style output
- **short-iso**: The default format augmented to show ISO 8601 wallclock timestamps.
- **short-monotonic**: The default format with monotonic timestamps.
- **short-precise**: The default format with microsecond precision
- **verbose**: Shows every journal field available for the entry, including those usually hidden internally.

These options allow you to display the journal entries in the whatever format best suits your current needs.

Active Process Monitoring

The `journalctl` command imitates how many administrators use `tail` for monitoring active or recent activity. This functionality is built into `journalctl`, allowing you to access these features without having to pipe to another tool.

Displaying Recent Logs

To display a set amount of records, you can use the `-n` option, which works exactly as `tail -n`.

By default, it will display the most recent 10 entries:

```
[jmedinar@localhost ~]$ journalctl -n
```

You can specify the number of entries you'd like to see with a number after the `-n`:

```
[jmedinar@localhost ~]$ journalctl -n 20
```

Following Logs

To actively follow the logs as they are being written, you can use the `-f` flag. Again, this works as you might expect if you have experience using `tail -f`:

```
[jmedinar@localhost ~]$ journalctl -f
```

Journal Maintenance

You may be wondering about the cost is of storing all of the data we've seen so far. Furthermore, you may be interesting in cleaning up some older logs and freeing up space.

Finding Current Disk Usage

You can find out the amount of space that the journal is currently occupying on disk by using the `--disk-usage` flag:

```
[jmedinar@localhost ~]$ journalctl --disk-usage
Journals take up 8.0M on disk.
```

Deleting Old Logs

If you wish to shrink your journal, you can do that in two different ways (available with `systemd` version 218 and later).

If you use the `--vacuum-size` option, you can shrink your journal by indicating a size. This will remove old entries until the total journal space taken up on disk is at the requested size:

```
[jmedinar@localhost ~]$ sudo journalctl --vacuum-size=1G
```

Another way that you can shrink the journal is providing a cutoff time with the `--vacuum-time` option. Any entries beyond that time are deleted. This allows you to keep the entries that have been created after a specific time.

For instance, to keep entries from the last year, you can type:

```
[jmedinar@localhost ~]$ sudo journalctl --vacuum-time=1years
```

Limiting Journal Expansion

You can configure your server to place limits on how much space the journal can take up. This can be done by editing the `/etc/systemd/journald.conf` file.

The following items can be used to limit the journal growth:

- **`SystemMaxUse=`**: Specifies the maximum disk space that can be used by the journal in persistent storage.
- **`SystemKeepFree=`**: Specifies the amount of space that the journal should leave free when adding journal entries to persistent storage.
- **`SystemMaxFileSize=`**: Controls how large individual journal files can grow to in persistent storage before being rotated.
- **`RuntimeMaxUse=`**: Specifies the maximum disk space that can be used in volatile storage (within the `/run` filesystem).
- **`RuntimeKeepFree=`**: Specifies the amount of space to be set aside for other uses when writing data to volatile storage (within the `/run` filesystem).
- **`RuntimeMaxFileSize=`**: Specifies the amount of space that an individual journal file can take up in volatile storage (within the `/run` filesystem) before being rotated.

By setting these values, you can control how `journald` consumes and preserves space on your server. Keep in mind that **`SystemMaxFileSize`** and **`RuntimeMaxFileSize`** will target archived files to reach stated limits. This is important to remember when interpreting file counts after a vacuuming operation.

Need more?

🎥 RHCSA 8 - Debugging with Journalctl