Lesson #3 - Software package management

Advanced Linux Administration

Karel Šrot

Motivation

- As a system administrator I need to be able to effectively provide and manage software installed on your systems in a lab.
- As a system administrator you need to be able to apply a defined software update strategy to meet security, policy or user requirements.

Learning objectives

- Understand how the software is being distributed
- Learn to manage installed software using dnf and rpm
- Understand software management challenges and obstacles
- Learn to create and manage yum/dnf software repositories
- Learn to rebuild RPM package from a source package (SRPM)
- Learn to install software in various packaging formats

Basic terminology

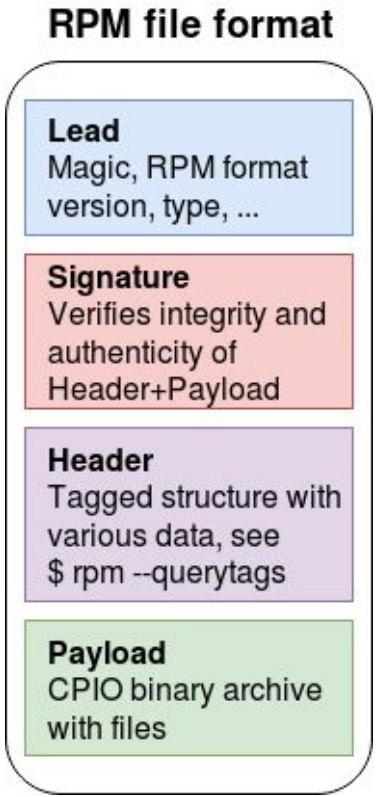
- **package manager** - a collection of software tools that automates the process of installing, upgrading, configuring, and removing computer programs for a computer's operating system in a consistent manner.
- **software package** - an archive file in a defined format containing files as well as necessary metadata for its deployment. Package metadata include package description, package version, dependencies, etc.
- **software repository** - a storage location from which software packages can be downloaded. Often contain also metadata about the individual packages.

Software management challenges

- Software installation and updates
- Security - having all security fixes installed
- Providing runtime environment for a specific (development/test/production) application
- Development vs Production
  - Developers prefer to have latest tools to utilize its benefits
  - Enterprise applications require stable and long-supported environment
- Missing software dependencies, conflicts, collisions
- Management of repositories and additional sources
- Coexistence of multiple software packaging solutions
- System upgrade/migration (to a different major version)
- Software asset management
- ...

Example: RPM file format

- Encapsulate either binary (RPM) and source (SRPM) package
- Checksums to verify integrity of the package
- GPG signature to prove authenticity of the package
- Filename format and naming convention
  - NAME-VERSION-RELEASE.ARCH.rpm, e.g. bash-5.0.7-1.fc30.x86_64.rpm
  - Subpackage name suffixes:
    - -devel, e.g. bash-devel-5.0.7-1.fc30.x86_64.rpm
    - -debuginfo, -debugsource
    - -doc, -libs, -common, -client, -server,...
  - Fedora naming guidelines



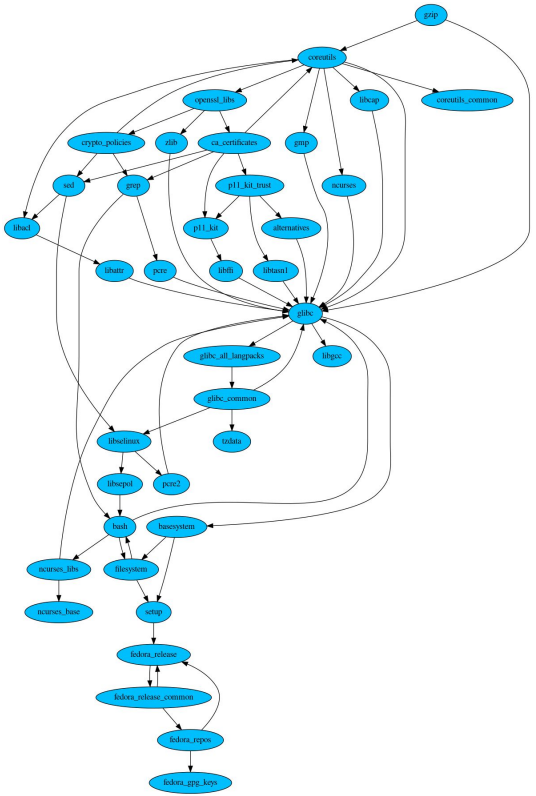
Example: RPM software package

```
$ file librepo-1.10.5-1.fc30.x86_64.rpm
librepo-1.10.5-1.fc30.x86_64.rpm: RPM v3.0 bin i386/x86_64 librepo-1.10.5-1.fc30
$ rpm2cpio librepo-1.10.5-1.fc30.x86_64.rpm | cpio -t
./usr/lib/.build-id
./usr/lib/.build-id/9b
./usr/lib/.build-id/9b/9cf60a479aef79589643d7b8614dfd451e5aea
./usr/lib64/librepo.so.0
./usr/share/doc/librepo
./usr/share/doc/librepo/README.md
./usr/share/licenses/librepo
./usr/share/licenses/librepo/COPYING
519 blocks
$ rpm -Kv librepo-1.10.5-1.fc30.x86_64.rpm
librepo-1.10.5-1.fc30.x86_64.rpm:
    Header V3 RSA/SHA256 Signature, key ID cfc659b9: OK
    Header SHA256 digest: OK
    Header SHA1 digest: OK
    Payload SHA256 digest: OK
    V3 RSA/SHA256 Signature, key ID cfc659b9: OK
    MD5 digest: OK
$
```

Example: gzip RPM package dependency graph

RPM package dependency graph

- Non-trivial even for a simple utility such as gzip
- Very complex considering all packages installed on a system or available in a repository/distribution
- *Dependency hell* issue arises around incompatible versions of shared packages or libraries on which several other packages have dependencies



Package managers in detail

Typical functions include:

- Looking up, downloading, installing or updating existing software from a software repository, removing installed software
- Ensuring the integrity and authenticity of the package by verifying their digital certificates and checksums
- Managing dependencies to ensure a package is installed with all packages it requires

Package manager in a Linux distribution

Common Linux distributions do typically

- Use one specific software package format for shipping software via distribution software repositories
- Use one (primary) package manager to (locally) manage software on the system
  - Fedora/OpenSuSe - rpm, Debian/Ubuntu - dpkg
- Provide additional package managers with advanced features working on top of the primary package manager or in parallel
  - Features like working with software repositories, automated dependency resolver, GUI, etc.
  - Fedora - dnf/PackageKit, OpenSuSe - Zypper/YaST, Debian - apt/aptitude/Synaptic

Overview of different package managers types

- Binary
- Source-code based
- Hybrid
- Distribution independent
- Application level dependency package managers

Binary package managers

Software is already built and ready for use. Different packages are distributed for given set of supported architectures.

Benefits:

- Easy to manage, faster package operations
- Improved software stability

Examples:

- rpm package manager / RPM package format
  - Fedora, CentOS, OpenSuSe, SLES
- dpkg package manager / DEB package format
  - Debian, Ubuntu

Source code-based package managers

Software is distributed in a source code and the user has to know how to compile it or it comes with a script automating the build process

Benefits:

- Access to the latest software
- Optimized compilation may provide performance increase

Examples:

- Apt-build / DEB (Debian, Ubuntu,...)
- ABS (Archlinux)

Hybrid package managers

Able to manage software distributed both in a source code form or as an already built binaries or a combination of both (building from source using some pre-compiled bits)

Examples:

- Portage/emerge (Gentoo, …)
- Nix package manager
- MacPorts

Distribution independent package managers

Unified package management for several Linux variants, using distribution independent/agnostic package format (bundling all the application dependencies)

Examples:

- Snap
- Flatpak
- AppImage

Application-level dependency package managers

Used for distribution of application specific software (programs, modules, libraries,...)

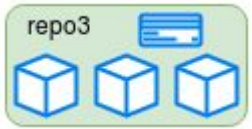
Can evoke problems or conflicts with content provided in the distribution natively.

Examples:

- CPAN (perl)
- pip (Python)
- RubyGems (Ruby)
- CTAN (TeX)

Software repository

- Storage location with a specific structure
- Hosts software packages and metadata utilized by software manager application
  - Minimal Fedora/RHEL/CentOS repository example




```
# tree localrepo/
localrepo/
├── Packages
│   ├── bash-4.4.19-10.el8.x86_64.rpm
│   └── createrepo_c-0.11.0-3.el8.x86_64.rpm
└── repodata
    ├── 35a9e6...fb232c-filelists.xml.gz
    ├── 4f160a...7a3e34-other.xml.gz
    ├── 727e7e...d7c49f-primary.xml.gz
    └── repomd.xml
```

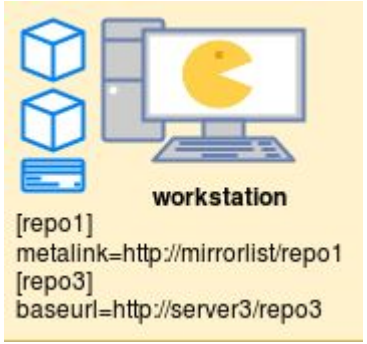
Software repository - cont.

- Can also serve additional content, e.g.
  - Files for the boot loader
  - Disk images, e.g. with installation media, for creating minimal boot media or system installation over the network (PXE)
  - Container images, …
  - Example: Fedora 38 mirror
- System can have multiple repositories configured (enabled/disabled)
- To access additional software a user may need to add 3rd party repositories
  - Fedora 3rd party repositories
  - Debian Unofficial repositories
- Repository metadata are being cached locally by package managers
  - `# dnf makecache` vs `# dnf clean all`
  - `# apt-get update` vs `# apt-get clean`

Example: Configuring Fedora/RHEL/CentOS repository

- On the system repositories are configured in `/etc/yum.repos.d/` directory

```
# cat /etc/yum.repos.d/localrepo.repo
[localrepo]
name=My Local Repo
baseurl=file:///tmp/localrepo
gpgcheck=0
```

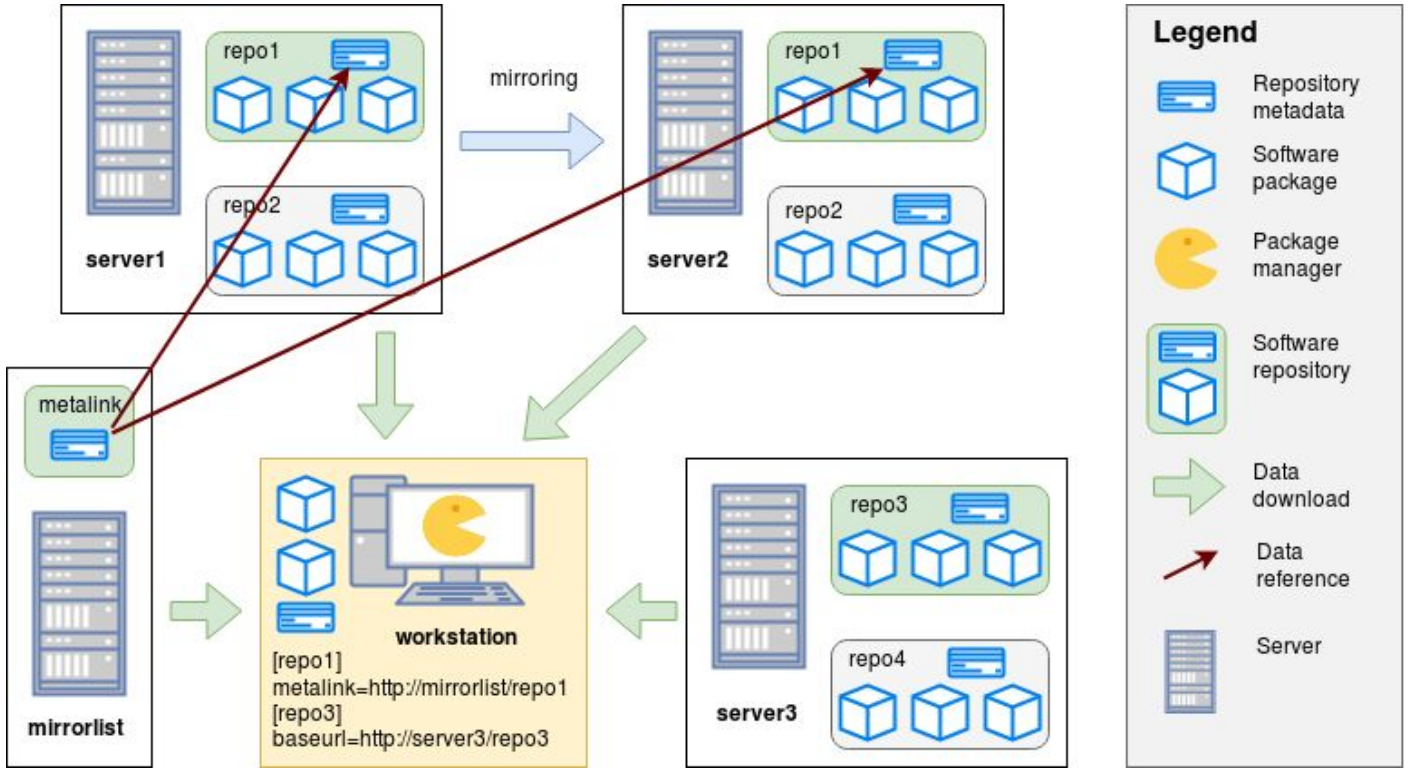


- Other common repository options:
  - `enabled`, `skip_if_unavailable`, `metalink`, `mirrorlist`, `sslcacert`, `sslverify`, `sslclientcert`, `sslclientkey`, `proxy`, `proxy_username`, `proxy_password` etc.
- Enable repository by editing the repo file or using `dnf`
  - `# dnf config-manager --set-enabled repositoryID`
  - Or temporarily using `--enablerepo`, e.g.
  - `# dnf --enablerepo repositoryID install pkgname`
- Repository is created (or modified) using `createrepo_c` command

Secure software distribution (Fedora/RHEL/Centos)

- Repository level
  - Provide HTTPS access using valid certificate
  - Sign individual RPMs using GPG key
  - (OPTIONALLY) Sign repository metadata using GPG key
- Repository configuration level
  - use HTTPS to access repositories
  - `sslverify=True, sslcacert, sslclientcert, sslclientkey`
  - `gpgcheck=True` (import appropriate gpgkeys to rpmdb)
  - (OPTIONALLY) `repo_gpgcheck=True` (in case repodata signatures are available)
- yum/dnf level
  - `gpgcheck=True` being a global default
- rpm command level
  - Option `pkgverify_level all`, verify using `rpm -Kv PACKAGE.rpm`
  - Rpm keys are imported with `rpm --import /etc/pki/rpm-gpg/key_file_name`

Example: Software distribution infrastructure



HowTo: Rebuild of RPM package from a SRPM package

- Download and install the SRPM (src.rpm) package

```
$ dnf download --source SOMEPKG
$ rpm -qlp SOMEPKG*src.rpm
$ rpm -i SOMEPKG*src.rpm
```

  Ignore printed warnings about non-existing user during package installation.
- Install tools/packages necessary for the build

```
$ sudo dnf -y install rpm-build
$ sudo dnf builddep -y ~/rpmbuild/SPECS/SOMEPKG.spec
```

- (optional) Do some tweaks in source files or a SPEC file

```
$ ls ~/rpmbuild/SOURCES
$ gedit ~/rpmbuild/SPECS/SOMEPKG.spec
```

- Rebuild the package

```
$ rpmbuild -bb ~/rpmbuild/SPECS/SOMEPKG.spec
```

Workshop

60 min

Workshop labs

- In the following next slides there are 10 labs total
  - 6 regular + 4 bonus
- Each lab has a time estimate how much time should you spend on it if everything goes well
- We encourage you to help each other or rise your hand to get help from lecturers
- If you couldn’t finish all labs during this class, you should complete them on your own later because learned skills will be used in following lectures or during a final practical exam.
- *HINT: focus on the lab content and leave any exploration or deep dive desires as a self-study for later*

Lab #1: `dnf` package manager (10 min)

Get familiar with `dnf` command so you can

- Search for packages in repositories (`dnf search ...`)
- Install a package (`dnf install ...`)
- Remove installed package (`dnf remove ...`)
- Check for available package updates (`dnf check-update`)
- Update installed packages (`dnf update …`)
- List enabled repositories (`dnf repolist ...`)
- List packages available in repositories (`dnf list ...`)
- Download RPM package from a repository (`dnf download …`)
- Query packages in repositories (`dnf repoquery --requires/--provides/--whatprovides ...`)

Resources: `# man dnf`

Lab #2: `rpm` package manager (10 min)

Get familiar with `rpm` command so you can

- Query details about an installed package or RPM file (`rpm -qi ...`)
- List installed packages (`rpm -qa ...`)
- List files within a package (`rpm -ql ...`, resp. `rpm -qlp ...`)
- Find out to which package a specific file belongs (`rpm -qf ...`)
- Verify checksums and signature of an RPM file (`rpm -Kv *.rpm`)
  - Modify the RPM file by appending some data to it and verify again
- Install downloaded RPM package (`rpm -iv ...`)
- Remove installed package (`rpm -e ...`)
- Query package Requires, Provides etc. (`rpm -q --requires ...`)

Resources:

- `# man rpm`

Lab #3: Rebuild and modify distribution package (10 min)

- Download Source RPM package for package `grep` and install it
- Observe the installed sources in `~/rpmbuild/SOURCES`
- Install all tools/packages necessary for rebuilding the `grep` package
- Rebuild the package using its SPEC file and observe the built RPMs
- Now modify the SPEC file by
  a. Adding `Provides: orange`
  b. Modifying the release to `Release: 99%{?dist}`
- Rebuild `grep` package RPMs again by using the modified SPEC file

Lab #4: Create a custom HTTP repository (10 min)

- Install `httpd` package and start the `httpd` service.
- In `/var/www/html/myrepo` directory create a repository with your custom built RPMs from the previous exercise and generate `repodata`
  - `# mkdir /var/www/html/myrepo`
  - Copy previously built grep package RPM file to `/var/www/html/myrepo`
  - `# createrepo  /var/www/html/myrepo`
  - Using `zcat` utility observe the content of *.xml.gz files in the `/var/www/html/myrepo/repodata` directory
- Configure enabled local repository by creating `/etc/yum.repos.d/my.repo` file with appropriate content.
  - baseurl should be specified as `baseurl=http://localhost/myrepo`
- Clean dnf cache and verify the repository is available and works as expected
  `# dnf list grep`

Resources: man createrepo_c, HowTo

Lab #5: Install debug packages with dnf (10 min)

Install gdb and -debuginfo packages to debug a binary using gdb command

- Having no coreutils-debuginfo/-debugsource packages installed, Issue gdb using following commands and observe the output.

```
$ gdb /usr/bin/sleep
Enable debuginfod for this session? (y or [n]) n    !!!!!!
(gdb) run 10
Ctrl+C
(gdb) list
(gdb) bt full
(gdb) quit
```

- Using dnf install all required debuginfo+debugsource packages required for debugging a binary (e.g. `/usr/bin/sleep`).
  - Hint: gdb tells you which packages are missing and suggests how to install them. Also make sure your system is updated.
- Repeat previous steps and compare the results

Lab #6: Managing Python modules using pip (10 min)

- Install `python3-pip`.
- Using `pip3` command install `numpy` module, show `numpy` module details.
- Install distribution python3-numpy package and find out which module is used

```
$ python3
>>> import numpy
>>> numpy.__path__
>>> numpy.__version__
```

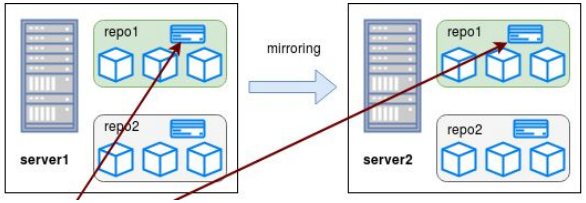
- Remove the installed numpy module using pip
- Having python3-numpy installed, try to install the module using pip again

Resources

- man pip
- pip documentation

Bonus Lab #7: Copy a HTTP repository with reposync (10 min)

- Using `reposync` create a local copy of remote HTTP(S) repository in `/tmp/copied`
- Ensure that repodata are available for the copied repository
- Configure the repository in `/etc/yum.repos.d/copied.repo`
- Verify the copied repository is available and works as expected



Resources

- `man dnf-reposync`, see Examples section

Bonus Lab #8: Installing application in Flatpak format (10 min)

- Install flatpak package and configure flathub repository (see docs below)
- Search Flathub page for Vim package and install it and run it using flatpak
- If running flatpak as a common user, better use the --user option

Resources

- Fedora Flatpak usage document
- Flathub homepage

Bonus Lab #9: Installing application in Snap format (10min)

- Following installation instruction for Fedora distribution in Snapcraft documentation Install and configure snapd
- Using snapd install and run hello-world package
  - Hint: Pay attention to the output of the `snap install` command

Resources

- Snapcraft documentation

Bonus Lab #10: Enable Copr repository (10 min)

Copr is designed to be a lightweight buildsystem that allows contributors to create packages, put them in repositories, and make it easy for users to install the packages onto their system

dnf has a Copr plugin to ease the work with Copr repositories. Learn how to use it.

- Following the Copr documentation enable `psss/edd` repository and install the edd package
- Uninstall the `edd` package and remove the repository

Resources

- `man dnf-copr`
- Copr documentation - how to enable repo