A Project Report on

Red Hat Package Manager for Payload Delivery

Submitted by
Minakshi Pushpendra Chavan (MIT21-A-03-PG-MTCST-34550 Roll No. 4)

Under the guidance of
Dr. B. S. Sonawane Sir

In partial fulfillment of the award of Master of Technology in Computer Science and Technology

Department of Computer Science and Technology,
G. S. Mandal's Maharashtra Institute of Technology,
(An Autonomous Institute)
Sambhaji Nagar (Formerly known as Aurangabad)
[2021-22]

DECLARATION

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included; I have adequately cited and referenced the original sources. I also declare that I have adhered to all the principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will cause disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper submission has not been taken when needed.

Place: Sambhaji Nagar (Formerly known as Aurangabad)
Date:

Signature and
Name of the Student

CERTIFICATE

This is to certify that the project - I report entitled "**Red Hat Package Manager for Payload Delivery**", submitted by **Minakshi Pushpendra Chavan** is the bonafide work completed under my supervision and guidance in partial fulfillment for the award of the Master of Technology in Computer Science and Technology, G. S. Mandal's Maharashtra Institute of Technology, An autonomous Institute from Sambhaji Nagar (Formerly known as Aurangabad), Maharashtra State.

Place: Sambhaji Nagar (Formerly known as Aurangabad)
Date:

Guide
Dr. B. S. Sonawane

Head of the Department
Dr. S. L. Kasar

Dr. S. P. Bhosle
Principal,
G. S. Mandal's Maharashtra Institute of Technology,
(An Autonomous Institute)
Sambhaji Nagar (Formerly known as Aurangabad)
Maharashtra State.

ACKNOWLEDGEMENT

I take this opportunity to express my deep and sincere profound gratitude to my guide, Prof. Dr. B. S. Sonawane, of G. S. Mandal's Maharashtra Institute of Technology, (An Autonomous Institute), Sambhaji Nagar (Formerly known as Aurangabad) for his unflagging support and continuous encouragement throughout the seminar work. Without his guidance and persistent help, this report would not have been possible.

I must acknowledge the facilities and staff of G. S. Mandal's Maharashtra Institute of Technology, Sambhaji Nagar.

It's my great pleasure to acknowledge my colleagues for providing constant support and encouragement. I am especially grateful to Sir Richard Stallman who ignited great minds across the globe and started the Free Software Movement which enabled the Linux enriched ecosystem throughout the digital world!

Minakshi Pushpendra Chavan,
M.Tech. CST - Student
Reg. No. MIT21-A-03-PG-MTCST-34550 - Roll No. 04

APPROVAL CERTIFICATE

This project - I report entitled "**Red Hat Package Manager for Payload Delivery**" by **Minakshi Pushpendra Chavan** is approved for Master of Technology in Computer Science and Technology, G. S. Mandal's Maharashtra Institute of Technology, (An Autonomous Institute), Sambhaji Nagar (Formerly known as Aurangabad), Maharashtra State.

Place: Sambhaji Nagar (Formerly known as Aurangabad)
Date:

Examiner: _____

Sign: _____

Name: _____

LIST OF ABBREVIATIONS

APT	Advanced Packaging Tool	**LZMA2**	Lempel–Ziv–Markov
AUR	Arch User Repository	**NIST**	National Institute of Standards & Technology
BTRFS	Butter File System	**NVR**	Name Version Release
DevOps	Development Operations	**OEM**	Original Equipment Manufacturer
DNF	Dandified YUM	**OS**	Operating System
DSS	Digital Signature Standard	**OTA**	Over the Air
ECC	Elliptic Curve Cryptography	**PoC**	Proof of Concept
FTP	File Transfer Protocol	**PPA**	Privacy Protection Act
GCC	GNU Compiler Collection	**RHEL**	Red Hat Enterprise Linux
GPG	GNU Privacy Guard	**RPM**	Red Hat Package Manager
HTTP	HyperText Transfer Protocol	**RSA**	Rivest-Shamir-Adelman
HTTPS	HyperText Transfer Protocol with SSL	**SPEC**	Specification File
IP	Internet Protocol	**UI**	User Interface
IPv6	Internet Protocol version 6	**URL**	Uniform Resource Locator
IT	Information Technology	**WAN**	Wide Area Network
KDE	K Desktop Environment	**YaST**	Yet Another Setup Group
LAN	Local Area Network	**YUM**	Yellowdog Updater Modifier

ABSTRACT

RPM Package Manager (RPM) (originally Red Hat Package Manager, now a recursive acronym) is a free and open-source package management system. The name RPM refers to the .rpm file format and the package manager program itself. RPM was intended primarily for Linux distributions; the file format is the baseline package format of the Linux Standard Base. Building an rpm is a complex process if the appropriate process isn't followed. This project talks about building an rpm to share the files over the internet and making sure that the package deployment not only takes care of appropriate version control but also integrity of the files deployed in the system.

For a system administrator performing software installation and maintenance, the use of package management rather than manual building has advantages such as simplicity, consistency and the ability for these processes to be automated and non-interactive. rpm uses Berkeley DB as the backend database although since 4.15 in 2019, it supports building rpm packages without Berkeley DB (–disable-bdb).

Features of RPM include:

- RPM packages can be cryptographically verified with GPG and MD5
- Original source archive(s) (e.g. .tar.gz, .tar.bz2) are included in SRPMs, making verification easier
- Delta update: PatchRPMs and DeltaRPMs, the RPM equivalent of a patch file, can incrementally update RPM-installed software
- Automatic build-time dependency evaluation.

The project delivers a mechanism to showcase a proof of concept from scratch to start the package build from zero to actual delivery after signing a package.

Keywords: rpm, package-management, database, gpg, md5, integrity, version-control

INDEX/CONTENTS

Chapter 1
INTRODUCTION

1.1 Introduction

Software used to be shipped pre-installed with hardware in the early days, and the deployment method was the embedded software in the Computer Chips. The idea of making software open was first brought in by Sir Richard Stallman in 1980's when the big corporations denied the source code distribution. Disappointed by the fact that his students won't be able to learn Software as they are supposed to be, Sir Richard Stallman started the free software foundation.

During the 1980s, the packages were directly compiled using source code and then binaries were created. A long journey from that day started and today, containers are used to deliver the packages and their updates over the air.

1.2 Prolog

Historically, software was provided either via FTP or mailing lists (this distribution would eventually grow to include basic websites). Only a few small files contained instructions to create a binary (usually in a tarfile). You would unpack the files, read the readme file, and if you had GCC or some other form of C compiler, you would typically run a ./configure script with some list of attributes such as path to library files, location to create new binaries, etc. Additionally, the configuration process would check your system for application dependencies. If any core requirements were missing, the configuration script would exit and you would not be able to proceed with the installation until all dependencies were met. If the configuration script completes successfully, a Makefile is created.

Once the Makefile existed, you would proceed to run the make command (this command is provided by whatever compiler you were using). The make command has a number of options called make flags that help optimize the resulting binaries for your system. In the earlier days of computing, this was very important as hardware struggled to keep up with the demands of modern software. Today, compilation options can be much more general, as most hardware is more than adequate for modern software.

Finally, after the make process is complete, you will need to run make install (or sudo make install) to actually install the software. As one can imagine, doing this for every single piece of software was time consuming and tedious – not to mention updating the software was a complicated and potentially very difficult process.

1.2.1 What is a package?

To combat this complexity, packages were invented. Packages collect multiple data files together into a single archive file for easier portability and storage, or simply compress files to reduce storage space. The binaries included in the package are precompiled to reasonable defaults chosen by the developer. Packages also contain metadata such as the name of the software, a description of its purpose, a version number, and a list of dependencies necessary for the software to function properly.

Several Linux variants have created their own package formats. Some of the most commonly used package formats include:

- **.deb**: This package format is used by Debian, Ubuntu, Linux Mint and several other derivatives. It was the first type of package to be created.
- **.rpm**: This package format was originally called Red Hat Package Manager. It is used by Red Hat, Fedora, SUSE and several other smaller distributions.
- **.tar.xz**: Although just a compressed tarball, this is the format used by Arch Linux.

Although packages themselves do not directly manage dependencies, they represented a huge step forward in Linux software management.

1.2.2 What is a Software Repository?

A few years ago, before smartphones became widespread, the idea of software storage was hard for many users to grasp unless they were involved in the Linux ecosystem. To this day, most Windows users still seem hard-wired to open a web browser to search for and install new software. However, those with smartphones have become accustomed to the idea of a software "store". The way smartphone users acquire software and the way package managers work are not dissimilar. Although there have been several attempts to create an attractive user interface for software repositories, the vast majority of Linux users still use the command line to install packages. Software repositories are a centralized list of all available software for any repository for which the system has been configured. Below are some examples of searching for a specific package in the repository (note that these have been abbreviated for brevity):

```
user@arch ~ $  aurman -Ss kate
extra/kate 18.04.2-2 (kde-applications kdebase)
    Advanced Text Editor
aur/kate-root 18.04.0-1 (11, 1.139399)
    Advanced Text Editor, patched to be able to run as root
```

```
aur/kate-git r15288.15d26a7-1 (1, 1e-06)
```

An advanced editor component which is used in numerous KDE applications requiring a text editing component

CentOS 7 using YUM

```
[user@centos ~]$ yum search kate
kate-devel.x86_64 : Development files for kate
kate-libs.x86_64 : Runtime files for kate
kate-part.x86_64 : Kate kpart plugin
Ubuntu using APT
user@ubuntu ~ $ apt search kate
Sorting... Done
Full Text Search... Done

kate/xenial 4:15.12.3-0ubuntu2 amd64
  powerful text editor

kate-data/xenial,xenial 4:4.14.3-0ubuntu4 all
  shared data files for Kate text editor

kate-dbg/xenial 4:15.12.3-0ubuntu2 amd64
  debugging symbols for Kate

kate5-data/xenial,xenial 4:15.12.3-0ubuntu2 all
  shared data files for Kate text editor
```

1.3 Keys and Certificates

Public and private keys form the basis of public key cryptography, also known as asymmetric cryptography. In public key cryptography, each public key matches only one private key. Together, they are used to encrypt and decrypt messages. If you encrypt a message using a person's public key, they can only decrypt it using their corresponding private key.

1.3.1 Example:

Bob wants to send Alice an encrypted email. To do this, Bob takes Alice's public key and encrypts his message to her. Then when Alice receives the message, she takes the private key that only she knows to decrypt the message from Bob.

Although attackers can try to compromise the server and read the message, they will not be able to because they do not have the private key to decrypt the message. Only Alice will be able to decrypt the message because she is the only one with the private key. And when Alice wants to reply, she simply repeats the process and encrypts her message to Bob using Bob's public key.

1.3.2 Difference between public key and private key

Public keys have been described by some as business addresses on the web – they are public and anyone can look them up and share them widely. In asymmetric encryption, public keys can be shared with everyone in the system. Once the sender has the public key, they use it to encrypt their message.

Each public key is paired with a unique private key. Think of a private key as a key to the front door of a company, where only you have a copy. This defines one of the main differences between the two types of keys. A private key ensures that only you can access the front door. For encrypted messages, you use this private key to decrypt the messages. Together, these keys help ensure the security of exchanged data. A message encrypted with a public key cannot be decrypted without using the corresponding private key.

1.3.3 Generating public and private keys

The public key and private key are not actually keys, but are really large prime numbers that are mathematically related. Relatedness in this case means that whatever is encrypted with a public key can only be decrypted with the associated private key.

A person cannot guess the private key based on knowledge of the public key. This makes the public key freely shareable. However, the private key belongs to only one person.
There are several well-known mathematical algorithms that are used to generate public and private keys. Some well-respected algorithms include:

1. **Rivest-Shamir-Adelman (RSA)** - the oldest of the public- and private-key cryptographic systems. It is often used to transfer shared keys for symmetric key cryptography
2. **Digital Signature Standard (DSS)** – Federal Information Processing Standard specifying algorithms that can be used to generate digital signatures used by NIST
3. **Elliptic Curve Cryptography (ECC)** – As the name suggests, ECC relies on elliptic curves to generate keys. Often used for key agreement and digital signatures.

1.3.4 Business benefits of public-private key encryption

By using a public and private key for encryption and decryption, recipients can be sure that the data is what the sender says it is. The recipient is assured of the confidentiality, integrity and authenticity of the data.

- **Confidentiality** is ensured because content that is secured with a public key can only be decrypted with a private key. This ensures that only the intended recipient can review the content at any time
- **Integrity** is ensured because part of the decryption process requires checking that the message received matches the message sent. This ensures that the message has not been changed in the meantime.
- **Authenticity** is ensured because every message Alice sends to Bob is also signed with Alice's private key. The only way to decrypt Alice's private key is with her public key, which Bob has access to. By signing a message with her private key, Alice ensures the authenticity of the message and shows that it really came from her.

Chapter 2
LITERATURE SURVEY

2.1 Package Delivery Methodologies.

What are the most used package managers? As suggested in the output above, package managers are used to interact with software repositories. The following is a brief overview of some of the most prominent package managers.

2.1.1 RPM-based package managers

Updating RPM-based systems, especially those based on Red Hat technologies, has a very interesting and detailed history. In fact, the current versions of yum (for enterprise distributions) and DNF (for the community) combine several open source projects to provide their current functionality.

Initially, Red Hat used a package manager called RPM (Red Hat Package Manager), which is still used today. However, its primary use is to install RPMs you have locally, not to search software repositories. A package manager called up2date was created to notify users of package updates and allow them to search remote repositories and install dependencies easily. While it served its purpose, some community members felt that up2date had some significant flaws.

The current yum incantation comes from several different community efforts. Yellowdog Updater (YUP) was developed in 1999-2001 by the folks at Terra Soft Solutions as a back-end engine for the Yellow Dog Linux graphical installer. Duke University liked the idea of YUP and decided to improve it. They created the Yellowdog Updater, Modified (yum), which was eventually modified to help manage university Red Hat Linux systems. Yum grew in popularity and it was estimated that by 2005 more than half of the Linux market would use it. Today, almost every Linux distribution that uses RPM uses yum for package management (with a few notable exceptions).

Working with yum

In order for yum to download and install packages from an Internet repository, the files must be located in /etc/yum.repos.d/ and have a .repo extension. Here is an example repo file:

```
[local_base]
name=Base CentOS  (local)
baseurl=http://7-repo.apps.home.local/yum-repo/7/
enabled=1
gpgcheck=0
```

This is for one of my local repositories, which explains why GPG checking is turned off. If this check were turned on, each package would have to be signed with a cryptographic key and the corresponding key would have to be imported into the system receiving the updates. Since I maintain this repository myself, I trust the packages and don't bother signing them.

Once the repository file is in place, you can start installing packages from the remote repository. The most basic command is yum update, which updates every currently installed package. This does not require a specific step to restore information about repositories; this is done automatically. A sample command is shown below:

```
[user@centos ~]$ sudo yum update
Loaded plugins: fastestmirror, product-id, search-disabled-repos,
subscription-manager
local_base                                | 3.6 kB  00:00:00
local_epel                                | 2.9 kB  00:00:00
local_rpm_forge                           | 1.9 kB  00:00:00
local_updates                             | 3.4 kB  00:00:00
spideroak-one-stable                      | 2.9 kB  00:00:00
zfs                                       | 2.9 kB  00:00:00
(1/6): local_base/group_gz                | 166 kB  00:00:00
(2/6): local_updates/primary_db           | 2.7 MB  00:00:00
(3/6): local_base/primary_db              | 5.9 MB  00:00:00
(4/6): spideroak-one-stable/primary_db    |  12 kB  00:00:00
(5/6): local_epel/primary_db              | 6.3 MB  00:00:00
(6/6): zfs/x86_64/primary_db              |  78 kB  00:00:00
local_rpm_forge/primary_db                | 125 kB  00:00:00
Determining fastest mirrors
Resolving Dependencies
--> Running transaction check
```

Working with Zypper

Zypper is another package manager meant to help manage RPMs. This package manager is most commonly associated with SUSE (and openSUSE) but has also seen adoption by MeeGo, Sailfish OS, and Tizen. It was originally introduced in 2006 and has been iterated upon ever since. There is not a whole lot to say other than Zypper is used as the back end for the system administration tool YaST and some users find it to be faster than yum.

Zypper's usage is very similar to that of yum. To search for, update, install or remove a package, simply use the following:

```
zypper search kate
zypper update
zypper install kate
zypper remove kate
```

Some major differences come into play in how repositories are added to the system with zypper. Unlike the package managers discussed above, zypper adds repositories using the package manager itself. The most common way is via a URL, but zypper also supports importing from repo files.

```
suse:~ # zypper addrepo
http://download.videolan.org/pub/vlc/SuSE/15.0 vlc
Adding repository 'vlc' [done]
Repository 'vlc' successfully added

Enabled     : Yes
Autorefresh : No
GPG Check   : Yes
URI         : http://download.videolan.org/pub/vlc/SuSE/15.0
Priority    : 99
You remove repositories in a similar manner:
suse:~ # zypper removerepo vlc
Removing repository 'vlc'
.................................[done]
Repository 'vlc' has been removed.
Use the zypper repos command to see what the status of
repositories are on your system:
suse:~ # zypper repos
Repository priorities are without effect. All enabled repositories
share the same priority.

#  | Alias                     | Name
| Enabled | GPG Check | Refresh
---+---------------------------+---------------------------------
-------+---------+-----------+--------
 1 | repo-debug                | openSUSE-Leap-15.0-Debug
```

```
| No      | ----      | ----
 2 | repo-debug-non-oss        | openSUSE-Leap-15.0-Debug-Non-Oss
| No      | ----      | ----
 3 | repo-debug-update         | openSUSE-Leap-15.0-Update-Debug
| No      | ----      | ----
 4 | repo-debug-update-non-oss |
openSUSE-Leap-15.0-Update-Debug-Non-Oss | No      | ----      |
----
 5 | repo-non-oss              | openSUSE-Leap-15.0-Non-Oss
| Yes     | ( p) Yes  | Yes
 6 | repo-oss                  | openSUSE-Leap-15.0-Oss
| Yes     | ( p) Yes  | Yes
```

zypper even has a similar ability to determine what package name contains files or binaries. Unlike YUM, it uses a hyphen in the command (although this method of searching is deprecated):

```
localhost:~ # zypper what-provides kate
Command 'what-provides' is replaced by 'search --provides
--match-exact'.
See 'help search' for all available options.
Loading repository data...
Reading installed packages...

S  | Name | Summary              | Type
---+------+----------------------+------------
i+ | Kate | Advanced Text Editor | application
i  | kate | Advanced Text Editor | package
```

As with YUM and DNF, Zypper has a much richer feature set than covered here.

2.1.2 Debian-based package managers

One of the oldest Linux distributions currently maintained, Debian's system is very similar to RPM-based systems. They use .deb packages, which can be managed by a tool called dpkg. dpkg is very similar to rpm in that it was designed to manage packages that are available locally. It does no dependency resolution (although it does dependency checking), and has no reliable way to interact with remote repositories. In order to improve the user experience and ease of use, the Debian project commissioned a project called Deity. This codename was eventually abandoned and changed to Advanced Package Tool (APT).

Released as test builds in 1998 (before making an appearance in Debian 2.1 in 1999), many users consider APT one of the defining features of Debian-based systems. It makes use of repositories in a similar fashion to RPM-based systems, but instead of individual .repo files that yum uses, apt has historically used /etc/apt/sources.list to manage repositories. More recently, it also ingests files from /etc/apt/sources.d/. Following the examples in the RPM-based package managers, to accomplish the same thing on Debian-based distributions you have a few options. You can edit/create the files manually in the aforementioned locations from the terminal, or in some cases, you can use a UI front end (such as Software & Updates provided by Ubuntu et al.). To provide the same treatment to all distributions, I will cover only the command-line options. To add a repository without directly editing a file, you can do something like this:

```
user@ubuntu:~$ sudo apt-add-repository "deb
http://APT.spideroak.com/ubuntu-spideroak-hardy/ release
restricted"
```

This will create a spideroakone.list file in /etc/apt/sources.list.d. Obviously, these lines change depending on the repository being added. If you are adding a Personal Package Archive (PPA), you can do this:

```
user@ubuntu:~$ sudo apt-add-repository ppa:gnome-desktop
```

NOTE: Debian does not support PPAs natively.
After a repository has been added, Debian-based systems need to be made aware that there is a new location to search for packages. This is done via the apt-get update command:

```
user@ubuntu:~$ sudo apt-get update
Get:1 http://security.ubuntu.com/ubuntu xenial-security InRelease
[107 kB]
Hit:2 http://APT.spideroak.com/ubuntu-spideroak-hardy release
InRelease
Hit:3 http://ca.archive.ubuntu.com/ubuntu xenial InRelease
Get:4 http://ca.archive.ubuntu.com/ubuntu xenial-updates InRelease
[109 kB]
Get:5 http://security.ubuntu.com/ubuntu xenial-security/main amd64
Packages [517 kB]
Get:6 http://security.ubuntu.com/ubuntu xenial-security/main i386
Packages [455 kB]
Get:7 http://security.ubuntu.com/ubuntu xenial-security/main
```

```
Translation-en [221 kB]
...

Fetched 6,399 kB in 3s (2,017 kB/s)
Reading package lists... Done
```

Now that the new repository is added and updated, you can search for a package using the apt-cache command:

```
user@ubuntu:~$ apt-cache search kate
aterm-ml - Afterstep XVT - a VT102 emulator for the X window
system
frescobaldi - Qt4 LilyPond sheet music editor
gitit - Wiki engine backed by a git or darcs filestore
jedit - Plugin-based editor for programmers
kate - powerful text editor
kate-data - shared data files for Kate text editor
kate-dbg - debugging symbols for Kate
katepart - embeddable text editor component
```

To install kate, simply run the corresponding install command:

```
user@ubuntu:~$ sudo apt-get install kate
```

2.1.3 Arch-based package managers

Arch Linux uses a package manager called pacman. Unlike .deb or .rpm files, pacman uses a more traditional tarball with the LZMA2 compression (.tar.xz). This enables Arch Linux packages to be much smaller than other forms of compressed archives (such as gzip). Initially released in 2002, pacman has been steadily iterated and improved. One of the major benefits of pacman is that it supports the Arch Build System, a system for building packages from source. The build system ingests a file called a PKGBUILD, which contains metadata (such as version numbers, revisions, dependencies, etc.) as well as a shell script with the required flags for compiling a package conforming to the Arch Linux requirements. The resulting binaries are then packaged into the aforementioned .tar.xz file for consumption by pacman.

This system led to the creation of the Arch User Repository (AUR) which is a community-driven repository containing PKGBUILD files and supporting patches or scripts. This allows for a virtually endless amount of software to be available in Arch. The obvious advantage of this system is that if a user (or maintainer) wishes to make software available to the public, they do not have to go through official channels to get it accepted in the main

repositories. The downside is that it relies on community curation similar to Docker Hub, Canonical's Snap packages, or other similar mechanisms. There are numerous AUR-specific package managers that can be used to download, compile, and install from the PKGBUILD files in the AUR.

Working with pacman and official repositories

Arch's main package manager, pacman, uses flags instead of command words like yum and apt. For example, to search for a package, you would use pacman -Ss. As with most commands on Linux, you can find both a manpage and inline help. Most of the commands for pacman use the sync (-S) flag. For example:

```
user@arch ~ $ pacman -Ss kate

extra/kate 18.04.2-2 (kde-applications kdebase)
    Advanced Text Editor
extra/libkate 0.4.1-6 [installed]
    A karaoke and text codec for embedding in ogg
extra/libtiger 0.3.4-5 [installed]
    A rendering library for Kate streams using Pango and Cairo
extra/ttf-cheapskate 2.0-12
    TTFonts collection from dustimo.com
community/haskell-cheapskate 0.1.1-100
    Experimental markdown processor.
```

Arch also uses repositories similar to other package managers. In the output above, search results are prefixed with the repository they are found in (extra/ and community/ in this case). Similar to both Red Hat and Debian-based systems, Arch relies on the user to add the repository information into a specific file. The location for these repositories is /etc/pacman.conf. The example below is fairly close to a stock system. I have enabled the [multilib] repository for Steam support:

```
[options]
Architecture = auto

Color
CheckSpace

SigLevel    = Required DatabaseOptional
LocalFileSigLevel = Optional
```

```
[core]
Include = /etc/pacman.d/mirrorlist

[extra]
Include = /etc/pacman.d/mirrorlist

[community]
Include = /etc/pacman.d/mirrorlist

[multilib]
Include = /etc/pacman.d/mirrorlist
```

It is possible to specify a specific URL in pacman.conf. This functionality can be used to make sure all packages come from a specific point in time. If, for example, a package has a bug that affects you severely and it has several dependencies, you can roll back to a specific point in time by adding a specific URL into your pacman.conf and then running the commands to downgrade the system:

```
[core]
Server=https://archive.archlinux.org/repos/2017/12/22/$repo/os/$ar
ch
```

Like Debian-based systems, Arch does not update its local repository information until you tell it to do so. You can refresh the package database by issuing the following command:

```
user@arch ~ $ sudo pacman -Sy

:: Synchronizing package databases...
 core
130.2 KiB   851K/s 00:00
[############################################################] 100%
 extra
1645.3 KiB  2.69M/s 00:01
[############################################################] 100%
 community
4.5 MiB  2.27M/s 00:02
[############################################################] 100%
 multilib is up to date
```

As you can see in the above output, pacman thinks that the multilib package database is up to date. You can force a refresh if you think this is incorrect by running pacman -Syy. If you want to update your entire system (excluding packages installed from the AUR), you can run pacman -Syu:

```
user@arch ~ $ sudo pacman -Syu

:: Synchronizing package databases...
 core is up to date
 extra is up to date
 community is up to date
 multilib is up to date
:: Starting full system upgrade...
resolving dependencies...
looking for conflicting packages...

Packages (45) ceph-13.2.0-2  ceph-libs-13.2.0-2
debootstrap-1.0.105-1  guile-2.2.4-1  harfbuzz-1.8.2-1
harfbuzz-icu-1.8.2-1  haskell-aeson-1.3.1.1-20
              haskell-attoparsec-0.13.2.2-24
haskell-tagged-0.8.6-1  imagemagick-7.0.8.4-1
lib32-harfbuzz-1.8.2-1  lib32-libgusb-0.3.0-1
lib32-systemd-239.0-1
              libgit2-1:0.27.2-1  libinput-1.11.2-1
libmagick-7.0.8.4-1  libmagick6-6.9.10.4-1  libopenshot-0.2.0-1
libopenshot-audio-0.1.6-1  libosinfo-1.2.0-1
              libxfce4util-4.13.2-1  minetest-0.4.17.1-1
minetest-common-0.4.17.1-1  mlt-6.10.0-1
mlt-python-bindings-6.10.0-1  ndctl-61.1-1  netctl-1.17-1
              nodejs-10.6.0-1

Total Download Size:      2.66 MiB
Total Installed Size:   879.15 MiB
Net Upgrade Size:      -365.27 MiB

:: Proceed with installation? [Y/n]
```

In the scenario mentioned earlier regarding downgrading a system, you can force a downgrade by issuing pacman -Syyuu. It is important to note that this should not be undertaken lightly. This should not cause a problem in most cases; however, there is a chance

that downgrading of a package or several packages will cause a cascading failure and leave your system in an inconsistent state. USE WITH CAUTION!
To install a package, simply use pacman -S kate:

```
user@arch ~ $ sudo pacman -S kate

resolving dependencies...
looking for conflicting packages...

Packages (7) editorconfig-core-c-0.12.2-1  kactivities-5.47.0-1
kparts-5.47.0-1  ktexteditor-5.47.0-2
syntax-highlighting-5.47.0-1  threadweaver-5.47.0-1
             kate-18.04.2-2

Total Download Size:   10.94 MiB
Total Installed Size:  38.91 MiB

:: Proceed with installation? [Y/n]
```

Chapter 3
SYSTEM DESIGN

3.1 RPM Specification

A SPEC file can be thought of as the "recipe" that the rpmbuild utility uses to actually build an RPM. It tells the build system what to do by defining instructions in a series of sections. The sections are defined in the Preamble and the Body. The Preamble contains a series of metadata items that are used in the Body. The Body contains the main part of the instructions.

3.1.1 Preamble Items

This table lists the items used in the Preamble section of the RPM SPEC file:

SPEC Directive	Definition
`Name`	The base name of the package, which should match the SPEC file name.
`Version`	The upstream version number of the software.
`Release`	The number of times this version of the software was released. Normally, set the initial value to 1%{?dist}, and increment it with each new release of the package. Reset to 1 when a new Version of the software is built.
`Summary`	A brief, one-line summary of the package.
`License`	The license of the software being packaged. For packages distributed in community distributions such as Fedora this must be an open source license abiding by the specific distribution's licensing guidelines.
`URL`	The full URL for more information about the program. Most often this is the upstream project website for the software being packaged.
`Source0`	Path or URL to the compressed archive of the upstream source code (unpatched, patches are handled elsewhere). This should point to an accessible and reliable storage of the archive, for example, the upstream page and not the packager's local storage. If needed, more SourceX directives can be added, incrementing the number each time, for example: Source1, Source2, Source3, and so on.
`Patch0`	The name of the first patch to apply to the source code if necessary. If needed, more PatchX directives can be added, incrementing the number each time, for example: Patch1, Patch2, Patch3, and so on.

`BuildArch`	If the package is not architecture dependent, for example, if written entirely in an interpreted programming language, set this to BuildArch: noarch. If not set, the package automatically inherits the Architecture of the machine on which it is built, for example x86_64.
`BuildRequires`	A comma- or whitespace-separated list of packages required for building the program written in a compiled language. There can be multiple entries of BuildRequires, each on its own line in the SPEC file.
`Requires`	A comma- or whitespace-separated list of packages required by the software to run once installed. There can be multiple entries of Requires, each on its own line in the SPEC file.
`ExcludeArch`	If a piece of software can not operate on a specific processor architecture, you can exclude that architecture here.

The Name, Version, and Release directives comprise the file name of the RPM package. RPM Package Maintainers and Systems Administrators often call these three directives N-V-R or NVR, because RPM package filenames have the NAME-VERSION-RELEASE format.

You can get an example of an NAME-VERSION-RELEASE by querying using rpm for a specific package:

```
$ rpm -q python
python-2.7.5-34.el7.x86_64
```

Here, python is the Package Name, 2.7.5 is the Version, and 34.el7 is the Release. The final marker is x86_64, which signals the architecture. Unlike the NVR, the architecture marker is not under direct control of the RPM packager, but is defined by the rpmbuild build environment. The exception to this is the architecture-independent noarch package.

3.1.2 Body Items

This table lists the items used in the Body section of the RPM SPEC file:

SPEC Directive	**Definition**
`%description`	A full description of the software packaged in the RPM. This description can span multiple lines and can be broken into paragraphs.
`%prep`	Command or series of commands to prepare the software to be built, for example, unpacking the archive in Source0. This directive can contain a shell script.

`%build`	Command or series of commands for actually building the software into machine code (for compiled languages) or byte code (for some interpreted languages).
`%install`	Command or series of commands for copying the desired build artifacts from the %builddir (where the build happens) to the %buildroot directory (which contains the directory structure with the files to be packaged). This usually means copying files from ~/rpmbuild/BUILD to ~/rpmbuild/BUILDROOT and creating the necessary directories in ~/rpmbuild/BUILDROOT. This is only run when creating a package, not when the end-user installs the package. See Working with SPEC files for details.
`%check`	Command or series of commands to test the software. This normally includes things such as unit tests.
`%files`	The list of files that will be installed in the end user's system.
`%changelog`	A record of changes that have happened to the package between different Version or Release builds.

3.1.3 Advanced items

The SPEC file can also contain advanced items. For example, a SPEC file can have scriptlets and triggers. They take effect at different points during the installation process on the end user's system (not the build process).

BuildRoots

In the context of RPM packaging, "buildroot" is a chroot environment. This means that the build artifacts are placed here using the same filesystem hierarchy as will be in the end user's system, with "buildroot" acting as the root directory. The placement of build artifacts should comply with the filesystem hierarchy standard of the end user's system.

The files in "buildroot" are later put into a cpio archive, which becomes the main part of the RPM. When RPM is installed on the end user's system, these files are extracted in the root directory, preserving the correct hierarchy.

RPM Macros

An rpm macro is a straight text substitution that can be conditionally assigned based on the optional evaluation of a statement when certain built-in functionality is used. What this means is that you can have RPM perform text substitutions for you so that you don't have to.

This is useful when, for example, referencing the packaged software Version multiple times in the SPEC file. You define Version only once - in the %{version} macro. Then use %{version} throughout the SPEC file. Every occurrence will be automatically substituted by Version you defined previously.

A common macro is %{?dist}, which signifies the “distribution tag”. It signals which distribution is used for the build.

For example:

```
# On a RHEL 7.x machine
$ rpm --eval %{?dist}
.el7

# On a Fedora 23 machine
$ rpm --eval %{?dist}
.fc23
```

3.2 RPM Building

You have created some software that you want to install on Red Hat Enterprise Linux systems. Now that it is done, the question is, “How do you gather up the software to make it easy for others to install and manage?”

The answer is to package it into an RPM. Although it is possible to just drop your software (via a tarball or another type of archive file) into a Linux system, packaging your Linux software as an RPM lets you:

- Include metadata with the package that describes its components, version number, size, package group, project URL, and many other pieces of information.
- Add the package to a yum repository so clients can easily find your software.
- Have clients use common Linux tools (yum, rpm, and PackageKit) to install, remove, and manage your software.
- Easily update and deploy new versions of the software, using the same Linux installation tools.

You do not have to be a programmer to create RPMs: you only need to understand how to create a SPEC file and use commands to build that SPEC file and package contents into an RPM. These procedures are outlined in this document. Building an RPM is not only useful for managing your company's software, but it is also listed as a skill that could be tested for on a Red Hat Certified Engineer (RHCE) exam.

3.2.1 UNDERSTANDING THE PROCESS OF BUILDING RPMS

The process of building an RPM requires knowing how to use a text editor and how to run a few commands to build, sign, and distribute the RPM. With your software in hand, most of the work needed to build the RPM involves creating a SPEC file. Within the SPEC file, you can:

- Identify the commands, configuration files, documentation, and other items in your package.
- Define where components are ultimately installed on the target Linux system.
- Set permissions and ownership of each file.
- Note when your package is dependent on other components being available.
- Tag files as configuration or documentation files.
- Have extra commands executed on the target system when the package is installed or uninstalled (such as creating user accounts, making directories, or moving files around).
- Add changelog entries to identify what changes have gone into each version of your software.

Once you have mastered the most critical features for building an RPM (as covered in this document), you will find that there is a wealth of features in RPM packaging tools to provide more powerful and flexible ways to create RPMs. For example, you can add platform-specific tags to an SPEC file so you can use the same file to build RPMs for multiple computer architectures.

3.2.2 REBUILDING AN EXISTING SOURCE CODE PACKAGE INTO AN RPM

The best way to learn how to create an RPM package is to start with an existing source code RPM package and rebuild it. Going through the process will let you see the procedures and components that go into building an RPM. This section outlines steps for rebuilding the tree RPM package from an existing source code package.

NOTE: Once you build this RPM, do not use it on a production system as the package will conflict with one already in your Red Hat Enterprise Linux software channels.

1. **Log In**: Log into a Red Hat Enterprise Linux system as a regular user (not root).
2. **Get a Source Code Package**: Download a working source code package. This example uses the tree source code package:

```
$ wget ftp://path/to/en/os/SRPMS/tree-1.5.3-2.el6.src.rpm
```

3. **Install the Source Code**: Install the source code (which should be in your current directory) into a new rpmbuild directory:

```
$ rpm -ihv tree-1.5.3-2.el6.src.rpm
```

This creates an rpmbuild directory structure in your home directory similar to the following:

```
~/SPECS
~/SPECS/tree.spec
~/BUILDROOT
~/SOURCES
~/SOURCES/tree-1.5.3.tgz
~/SOURCES/tree-1.2-no-strip.patch
~/SOURCES/tree-no-color-by-default.patch
~/SOURCES/tree-1.2-carrot.patch
~/SOURCES/tree-preserve-timestamps.patch
```

Notice that the new rpmbuild directory in your home directory includes a SPECS directory (which includes the tree.spec file) and a SOURCES directory. The SOURCES directory includes the tree-1.5.3.tgz tarball of the code and four patch files.

4. **Edit the Spec File**: Review (and possibly change) the spec file. Using either the vim or emacs editor will add color to the file as you edit it: vim ~/rpmbuild/SPECS/tree.spec. An example of that file is shown below. Save and exit the file after you are done changing it.

```
Summary: File system tree viewer
Name: tree
Version: 1.5.3
Release: 2%{?dist}
Group: Applications/File
License: GPLv2+
Url: http://mama.indstate.edu/users/ice/tree/
Source: ftp://mama.indstate.edu/linux/tree/tree-%{version}.tgz
Patch1: tree-1.2-carrot.patch
Patch2: tree-1.2-no-strip.patch
Patch3: tree-preserve-timestamps.patch
Patch4: tree-no-color-by-default.patch
BuildRoot:
%{_tmppath}/%{name}-%{version}-%{release}-root-%(%{__id_u} -n)
%description
The tree utility recursively displays the contents of directories in
a tree-like format. Tree is basically a UNIX port of the DOS tree
utility.%prep
%setup -q
# Fixed spelling mistake in man page.
%patch1 -p1 -b .carrot
```

```
# Don't strip binary in the Makefile -- let rpmbuild do it.
%patch2 -p1 -b .no-strip
# Preserve timestamp on man page.
%patch3 -p1 -b .preserve-timestamps
# Disable color output by default.
%patch4 -p1 -b .no-color-by-default
%build
make    CFLAGS="$RPM_OPT_FLAGS"    "CPPFLAGS=$(getconf    LFS_CFLAGS)"
%{?_smp_mlags}
%install
rm -rf $RPM_BUILD_ROOT
mkdir -p $RPM_BUILD_ROOT%{_bindir}
make
BINDIR=$RPM_BUILD_ROOT%{_bindir} \
MANDIR=$RPM_BUILD_ROOT%{_mandir}/man1 \
install
chmod -x $RPM_BUILD_ROOT%{_mandir}/man1/tree.1
%clean
rm -rf $RPM_BUILD_ROOT
%files
%defattr(-,root,root)
%{_bindir}/tree
%{_mandir}/man1/tree.1*
%doc README LICENSE
%changelog
...
```

After you install the rpm-build package, you can read about features of SPEC files in the /usr/share/doc/rpm-build*/spec file. The Name is the base name of the package. The Summary is a one-line description of the package. Version is the upstream version number on the package, while Release is the number you add as the packager to reflect multiple builds of the same upstream version (such as for bug fixes).

The URL points to the project site that produced the source code, and Source points to where the original source code used to make the package came from. BuildRoot identifies the location of the temporary directory where the RPM will be built. Other lines prepare the build environment, add patches, compile and build the software, identify the files and permissions in the package, and allow you to keep a log of the changes over time.

At the end of this document, you can find some rpm -qp options that you can use to check the content of the package you build.

5. **Build the RPM**: Use the rpmbuild command to turn your spec file and content into the RPM package for distribution. You can also package the source code into a separate source RPM (src.rpm). Install the rpm-build package (as root) and run rpmbuild (from your regular user account):

```
$ sudo yum install rpm-build
$ rpmbuild -ba ~/rpmbuild/SPECS/tree.spec
```

This results in a binary RPM and a source RPM in the RPMS and SRPMS subdirectories, respectively.

6. **Sign the RPM**: Signing an RPM requires that you create a public and private key pair, use the private key to sign your RPM, and then distribute the public key to clients so they can use that key to check your signed package.

```
$ gpg --gen-key
```

Generate public/private keys When you generate your public/private keys, you can use most of the defaults. The end of the output will be similar to the following:

```
pub   2048R/99A9CF07 2011-09-16
Key   fingerprint = 90BF B5DC 628E C9E0 88D0 E5D1 E828 4641 99A9
CF07
uid   Chris Negus (My own build of the tree package.)
<cnegus@redhat.com>
sub   2048R/48E60E56 2011-09-16
```

Use the key ID generated (in this case, 99A9CF07) to export your private key to a public key:

```
$ gpg -a -o RPM-GPG-KEY-ABC --export 99A9CF07
```

To make sure the key ID is used to sign your package, add a _gpg_name line to the .rpmmacros file in your home directory:

```
$ vi ~/.rpmmacros
%_gpg_name 99A9CF07
```

Now you are ready to sign the package:

```
$ rpm --resign  ~/rpmbuild/RPMS/x86_64/tree-1.5.3-2.el6.x86_64.rpm
```

7. **Publish the RPM in a yum Repository**: One way to make your RPM accessible is to create a yum repository that is accessible from your web server. Assuming a web server is running on the system on which you build your RPM, these steps publish the RPM and make a yum repository:

```
# mkdir /var/www/html/abc
# cp ~/RPM-GPG-KEY-ABC /var/www/html/abc/
```

Make the public key available

```
# cp ~/rpmbuild/RPMS/x86_84/tree-1.5.3-2.el6.x86_64.rpm
/var/www/html/abc/
# createrepo /var/www/html/abc
```

8. **Create a Repository (.repo) File**: Create a .repo file that identifies the URL to the repository. Clients that want to install the package will be able to simply copy the abc.repo file to their own RHEL system's /etc/yum.repos.d directory to enable it. Replace whatever.example.com with the FQDN of your own web server:

```
$ vim abc.repo
[abc-repo]
name=My ABC yum repository
baseurl=http://whatever.example.com/abc
gpgkey= http://whatever.example.com/RPM-GPG-KEY-ABC
$ cp abc.repo /var/www/html/abc
```

9. **Prepare Clients to Install the RPM**: To install your RPM, clients can simply copy your .repo file to their systems, then use the yum command to install any package from your repository:

```
# wget http://whatever.example.com/abc/abc.repo -O
/etc/yum.repos.d/abc.repo
# yum install tree
```

10. To update your RPM in the future, you can simply rebuild the RPM, copy the latest version to your yum repository directory, and rerun the createrepo command. Clients will get the new RPM the next time they install or update the package.

3.2.3 CHECKING YOUR RPM PACKAGE

Once you have finished building your RPM, you can use the rpm command to check its contents and make sure the signature worked properly. You can do this on any Red Hat Enterprise Linux system, as long as you can get the package and the public key. Start by importing the key used to sign the package and checking the signature:

```
# rpm --import ~/RPM-GPG-KEY-ABC
$ rpm -K ~/rpmbuild/RPMS/x86_64/tree-1.5.3-2.el6.x86_64.rpm Check
signature
~/rpmbuild/RPMS/x86_64/tree-1.5.3-2.el6.x86_64.rpm: sha1 md5 OK
```

3.2.4 BUILDING AN RPM FROM SCRATCH

To create your own RPM, you need to create your own spec file (and put it in the SPECS directory) and gather into a tarball the executables, scripts, user documentation files, and configuration files you want included in the RPM. You can create your spec file by simply copying an existing spec file and modifying it. As an alternative, you can use the vim or emacs command to open any new file that ends in .spec. The editor will automatically create a template within the new file for writing an RPM spec file. You can then follow the rest of the procedure described earlier in this document. To look at an example of a tarball of content for an RPM, try untarring the tree tarball included in the tree source code package:

```
$ tar xvf ~/rpmbuild/SOURCES/tree-1.5.3.tgz
tree-1.5.3/CHANGES
tree-1.5.3/INSTALL
tree-1.5.3/LICENSE
tree-1.5.3/Makefile
tree-1.5.3/README
tree-1.5.3/tree.c
tree-1.5.3/strverscmp.c
tree-1.5.3/man/tree.1
tree-1.5.3/man/tree.1.fr
```

To create your own tarball, you can simply put your content in a directory (such as ~/abc-1.0) and gather it into a tarball that is placed into the SOURCES directory:

```
$ tar -cvzf ~/rpmbuild/SOURCES/abc-1.0-1.tar.gz ~/abc-1.0/
```

Chapter 4
IMPLEMENTATION

4.1 Build an rpm spec and source.

```
[root@localhost ~]# yum install rpm-build rpm-sign -y
[root@localhost ~]# cat rpmbuild/SPECS/rpmdbrc.spec
Summary: The utility to identify rpm database corruption and
details around it.
Name: rpmdbrc
Version: 1.0.0
Release: 0.el7
Group: Applications/File
Source: rpmdbrc-%{version}.tar.gz
BuildArch: noarch
Requires: bash
Requires: rpm
License: GPLv2+
Vendor: Test Project
Packager: Minakshi Pushpendra Chavan
BuildRoot:  %{_tmppath}/%{name}-buildroot

%description
This Package provides the utility to check rpm database
corruption, fixes it and gives some more details around rpm
database.

%prep
%setup -q

%build

%install
rm -rf $RPM_BUILD_ROOT
install -m 0755 -d  $RPM_BUILD_ROOT/usr/sbin
install -p -m 755 usr/sbin/rpmdbrc
$RPM_BUILD_ROOT/usr/sbin/rpmdbrc
install -m 0755 -d  $RPM_BUILD_ROOT/usr/share/man/man1
install -p -m 644 usr/local/man/man8/rpmdbrc.1.gz
$RPM_BUILD_ROOT/usr/share/man/man1/rpmdbrc.1.gz
```

```
%clean
rm -rf $RPM_BUILD_ROOT

%files
%defattr(-,root,root)
/usr/sbin/rpmdbrc
%doc /usr/share/man/man1/rpmdbrc.1.gz

%post
echo "Please run "rpmdbrc --help" to know more about it!"

%postun

echo "Hope You liked this mini project!"

%changelog
* Sun Jul 21 Minakshi Pushpendra Chavan 1.0.0
- Initial version
```

```
[root@localhost ~]# tar tvf
/root/rpmbuild/SOURCES/rpmdbrc-1.0.0.tar.gz
drwxr-xr-x root/root         0 2017-10-29 11:43 rpmdbrc-1.0.0/
drwxr-xr-x root/root         0 2017-10-29 11:43 rpmdbrc-1.0.0/usr/
drwxr-xr-x root/root         0 2017-10-29 11:34
rpmdbrc-1.0.0/usr/local/
drwxr-xr-x root/root         0 2017-10-29 11:34
rpmdbrc-1.0.0/usr/local/man/
drwxr-xr-x root/root         0 2017-10-29 11:48
rpmdbrc-1.0.0/usr/local/man/man8/
-rw-r--r-- root/root       744 2017-10-29 11:48
rpmdbrc-1.0.0/usr/local/man/man8/rpmdbrc.1.gz
drwxr-xr-x root/root         0 2017-10-29 11:46
rpmdbrc-1.0.0/usr/sbin/
-rw-r--r-- root/root      8781 2017-10-29 11:43
rpmdbrc-1.0.0/usr/sbin/rpmdbrc

[root@localhost ~]# rpmbuild -ba rpmbuild/SPECS/rpmdbrc.spec
Executing(%prep): /bin/sh -e /var/tmp/rpm-tmp.ZQ05qa
```

```
+ umask 022
+ cd /root/rpmbuild/BUILD
+ cd /root/rpmbuild/BUILD
+ rm -rf rpmdbrc-1.0.0
+ /usr/bin/gzip -dc /root/rpmbuild/SOURCES/rpmdbrc-1.0.0.tar.gz
+ /usr/bin/tar -xf -
+ STATUS=0
+ '[' 0 -ne 0 ']'
+ cd rpmdbrc-1.0.0
+ /usr/bin/chmod -Rf a+rX,u+w,g-w,o-w .
+ exit 0
Executing(%build): /bin/sh -e /var/tmp/rpm-tmp.T15D7Q
+ umask 022
+ cd /root/rpmbuild/BUILD
+ cd rpmdbrc-1.0.0
+ exit 0
Executing(%install): /bin/sh -e /var/tmp/rpm-tmp.DdFwOx
+ umask 022
+ cd /root/rpmbuild/BUILD
+ '[' /root/rpmbuild/BUILDROOT/rpmdbrc-1.0.0-0.el7.x86_64 '!=' /
']'
+ rm -rf /root/rpmbuild/BUILDROOT/rpmdbrc-1.0.0-0.el7.x86_64
++ dirname /root/rpmbuild/BUILDROOT/rpmdbrc-1.0.0-0.el7.x86_64
+ mkdir -p /root/rpmbuild/BUILDROOT
+ mkdir /root/rpmbuild/BUILDROOT/rpmdbrc-1.0.0-0.el7.x86_64
+ cd rpmdbrc-1.0.0
+ rm -rf /root/rpmbuild/BUILDROOT/rpmdbrc-1.0.0-0.el7.x86_64
+ install -m 0755 -d
/root/rpmbuild/BUILDROOT/rpmdbrc-1.0.0-0.el7.x86_64/usr/sbin
+ install -p -m 755 usr/sbin/rpmdbrc
/root/rpmbuild/BUILDROOT/rpmdbrc-1.0.0-0.el7.x86_64/usr/sbin/rpmdb
rc
+ install -m 0755 -d
/root/rpmbuild/BUILDROOT/rpmdbrc-1.0.0-0.el7.x86_64/usr/share/man/
man1
+ install -p -m 644 usr/local/man/man8/rpmdbrc.1.gz
/root/rpmbuild/BUILDROOT/rpmdbrc-1.0.0-0.el7.x86_64/usr/share/man/
man1/rpmdbrc.1.gz
+ /usr/lib/rpm/find-debuginfo.sh --strict-build-id -m --run-dwz
--dwz-low-mem-die-limit 10000000 --dwz-max-die-limit 110000000
```

```
/root/rpmbuild/BUILD/rpmdbrc-1.0.0
/usr/lib/rpm/sepdebugcrcfix: Updated 0 CRC32s, 0 CRC32s did match.
+ /usr/lib/rpm/check-buildroot
+ /usr/lib/rpm/redhat/brp-compress
+ /usr/lib/rpm/redhat/brp-strip-static-archive /usr/bin/strip
+ /usr/lib/rpm/brp-python-bytecompile /usr/bin/python 1
+ /usr/lib/rpm/redhat/brp-python-hardlink
+ /usr/lib/rpm/redhat/brp-java-repack-jars
Processing files: rpmdbrc-1.0.0-0.el7.noarch
Provides: rpmdbrc = 1.0.0-0.el7
Requires(interp): /bin/sh /bin/sh
Requires(rpmlib): rpmlib(CompressedFileNames) <= 3.0.4-1
rpmlib(FileDigests) <= 4.6.0-1 rpmlib(PayloadFilesHavePrefix) <=
4.0-1
Requires(post): /bin/sh
Requires(postun): /bin/sh
Checking for unpackaged file(s): /usr/lib/rpm/check-files
/root/rpmbuild/BUILDROOT/rpmdbrc-1.0.0-0.el7.x86_64
Wrote: /root/rpmbuild/SRPMS/rpmdbrc-1.0.0-0.el7.src.rpm
Wrote: /root/rpmbuild/RPMS/noarch/rpmdbrc-1.0.0-0.el7.noarch.rpm
Executing(%clean): /bin/sh -e /var/tmp/rpm-tmp.xQPqSC
+ umask 022
+ cd /root/rpmbuild/BUILD
+ cd rpmdbrc-1.0.0
+ rm -rf /root/rpmbuild/BUILDROOT/rpmdbrc-1.0.0-0.el7.x86_64
+ exit 0
```

4.2. The built rpm is unsigned at this moment.

```
[root@localhost ~]# rpm -qpl
/root/rpmbuild/RPMS/noarch/rpmdbrc-1.0.0-0.el7.noarch.rpm
/usr/sbin/rpmdbrc
/usr/share/man/man1/rpmdbrc.1.gz
[root@localhost ~]# rpm -qpi
/root/rpmbuild/RPMS/noarch/rpmdbrc-1.0.0-0.el7.noarch.rpm
Name        : rpmdbrc
Version     : 1.0.0
Release     : 0.el7
Architecture: noarch
```

```
Install Date: (not installed)
Group       : Applications/File
Size        : 9515
License     : GPLv2+
Signature   : (none)
Source RPM  : rpmdbrc-1.0.0-0.el7.src.rpm
Build Date  : Sun 29 Oct 2017 11:52:57 AM MDT
Build Host  : localhost
Relocations : (not relocatable)
Packager    : Minakshi Pushpendra Chavan
Vendor      : Test Project
Summary     : The utility to identify rpm database corruption and
details around it.
Description :
This Package provides the utility to check rpm database
corruption, fixes it and gives some more details around rpm
database.
```

4.3 Generate a gpg keypair

```
[root@localhost ~]# gpg --gen-key
gpg (GnuPG) 2.0.22; Copyright (C) 2013 Free Software Foundation,
Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: directory `/root/.gnupg' created
gpg: new configuration file `/root/.gnupg/gpg.conf' created
gpg: WARNING: options in `/root/.gnupg/gpg.conf' are not yet
active during this run
gpg: keyring `/root/.gnupg/secring.gpg' created
gpg: keyring `/root/.gnupg/pubring.gpg' created
Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection?
RSA keys may be between 1024 and 4096 bits long.
```

```
What keysize do you want? (2048)
Requested keysize is 2048 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0)
Key does not expire at all
Is this correct? (y/N)
Key is valid for? (0)
Key does not expire at all
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: Minakshi Chavan
Email address: shenguleminakshi265@gmail.com
Comment: Test Comment
You selected this USER-ID:
    "Minakshi Chavan <shenguleminakshi265@gmail.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
You need a Passphrase to protect your secret key.

We need to generate a lot of random bytes. It is a good idea to
perform
some other action (type on the keyboard, move the mouse, utilize
the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to
perform
some other action (type on the keyboard, move the mouse, utilize
the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: /root/.gnupg/trustdb.gpg: trustdb created
gpg: key 8FCC8421 marked as ultimately trusted
```

```
public and secret key created and signed.

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f,
1u
pub   2048R/8FCC8421 2017-10-29
      Key fingerprint = 95ED 76E7 EC32 06B8 5B50  CDC3 C571 4953
8FCC 8421
uid                  Minakshi Chavan
<shenguleminakshi265@gmail.com>
sub   2048R/0D5521D2 2017-10-29


NOTE: The password used here is RedHat1!

[root@localhost ~]# gpg --list-keys
/root/.gnupg/pubring.gpg
------------------------
pub   2048R/8FCC8421 2017-10-29
uid                  Minakshi Chavan
<shenguleminakshi265@gmail.com>
sub   2048R/0D5521D2 2017-10-29
```

4.4 Export the public key.

```
[root@localhost ~]# gpg --export -a 'Minakshi Chavan' >
rpmdbrc-public-key.txt
[root@localhost ~]# cat rpmdbrc-public-key.txt
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2.0.22 (GNU/Linux)

mQENBFn2FRQBCADQl2EiM2wr63MWbLqSZ4giOYzhDd8MLrV6dWbU1AIBZ4RlauvX
/Fa23MHUGmIO4n9JOpA5FdOZmBG9rnsHsF15c102Lp/6s7e2M0JGZMJ4tYL5uVgY
UM0TClwNRFTloPkKzpd9aDwEErZYnuWc/xuCEcOiFTh2e2YYrDGc2CDt8nsWQi5g
uRm34oL2XEcDY1N3tYoLDDdj8BerZYjkA7dGUfKXaImlJI0wmnN379MqPZSuV+Zz
tV8bRyZgGUUhKXYhOZXsg9YvU2bgRhWVhfKL+MQblrj5m/J2zMtCroD7HkwxqT6f
c3RQeX03T3f3do1RBi/9kXQU+k7C7t++5pb5ABEBAAG0MlNuZWhhIEpvc2hpIChS
ZWQgSGF0IEludGVybikgPHNuZWpvc2hpQHJlZGhhdC5jb20+iQE5BBMBAgAjBQJZ
```

```
9hUUAhsDBwsJCAcDAgEGFQgCCQoLBBYCAwECHgECF4AACgkQxXFJU4/MhCFkxAf8
D9iGfBwo7xuVOFcMoCJ4hQKgTo2bSXD3tWW15wyeOiUUlHRl1weN4LqHKZeEkUHP
Wn8GPAjHbTI5tgXdh2qYD/R8j4cSYi1C0prJS0xjuroEUDK9YrfR9T2vHmX5/aTU
OwlmgSDAbRHsKSsIDhsucXtk5xs37wC12gqnxamMny9fiBYM7bZJ2zeJRNjy+bQf
1c8Q4FVEWl6EHIHBcmJXZ5Hl7Iwg45QPksRjUbiEWPRttsqGOoo9sQpFpfyUzi00
j9WRBjmfL9HzhykrfiLit+SpVbgBmxbqPmv7Tet9KX3VAGra4wJiqxzRsCjbqKNl
2lLTTlL80iAAKmJU7j/ZobkBDQRZ9hUUAQgApaWjp00Q5/wfQNh3nBtKXKWKFrur
N4LbvymuRQfpcvhPi+c0gq82NMaTlyZ+me/ZCWiPL5+0089rLHwdTmUCMX2+Jb+z
lApnoX1esZu4lgzKgBiz0f7VE9eSkYQLTtLu6SsvQJeQQ5Go1ObUmFkdE10aGgrv
XD+IzKIPSqiujHNpwIFfdsv1Le1QKIHogYzMnK39X35Y7hqCkdtw5Ll0RRz2E2zq
/1ZytgaJCPyU+4f2f0Qpv4x9DpYT08p0Y73VgWadlmkqJnUQeW3MccMw45R/23KW
M3+Jkz9HH1bw77RfUK1Aks+aIusLRnh4ND5+HQxDIxhOU3HHLVdBp/z2PQARAQAB
iQEfBBgBAgAJBQJZ9hUUAhsMAAoJEMVxSVOPzIQhk3sH/26BrS/yu0kowUnk2qKF
wO2jZ5FCy5KcKSGqHIxcP6Z0nnO1lGdJcq1UtASSypRHwz3Rz8PxFqkq7TNIUDfo
8+JKjWOfBsrHjK3Gjt48Rv6r8QDStWxWPCuZDVgPHyFA+Fn84011CwNYdi3ZA7L1
pVZR8/b2WL5SAjOVho9gMSjNZ4HOllENVbhx2KDu+kWPTaCCG+ix8iro377i1hzp
Nwp8H33YRGcAeOSq+vLCN9bPVAQpzbEhBDSM7XGk5n1fjEhrTvD/sdC7ql0HfCDK
fT8gLo4oy4JlFfZz353rJBPRX0RtgiWG9tuhj0E2qGxmLg5O0KWey9M/kgMhFELD
S2E=
=+Xyd
-----END PGP PUBLIC KEY BLOCK-----
```

4.5 Check the system's gpg public keys and import key.

```
[root@localhost ~]# rpm -q gpg-pubkey --qf
'%{name}-%{version}-%{release} --> %{summary}\n'
gpg-pubkey-fd431d51-4ae0493b --> gpg(Red Hat, Inc. (release key 2)
<security@redhat.com>)
gpg-pubkey-2fa658e0-45700c69 --> gpg(Red Hat, Inc. (auxiliary key)
<security@redhat.com>)

[root@localhost ~]# rpm --import rpmdbrc-public-key.txt
[root@localhost ~]# rpm -q gpg-pubkey --qf
'%{name}-%{version}-%{release} --> %{summary}\n'
gpg-pubkey-fd431d51-4ae0493b --> gpg(Red Hat, Inc. (release key 2)
<security@redhat.com>)
gpg-pubkey-2fa658e0-45700c69 --> gpg(Red Hat, Inc. (auxiliary key)
<security@redhat.com>)
gpg-pubkey-8fcc8421-59f61514 --> gpg(Minakshi Chavan
```

```
<shenguleminakshi265@gmail.com>)
```

4.6 Create an rpmmacros file for rpm signing.

```
[root@localhost ~]# vi .rpmmacros
[root@localhost ~]# cat .rpmmacros
%_signature gpg
%_gpg_name Minakshi Chavan
```

4.7 Sign the rpm

```
[root@localhost ~]# rpm --addsign
/root/rpmbuild/RPMS/noarch/rpmdbrc-1.0.0-0.el7.noarch.rpm
Enter pass phrase:
Pass phrase is good.
/root/rpmbuild/RPMS/noarch/rpmdbrc-1.0.0-0.el7.noarch.rpm:

[root@localhost ~]# rpm -qip
/root/rpmbuild/RPMS/noarch/rpmdbrc-1.0.0-0.el7.noarch.rpm
Name        : rpmdbrc
Version     : 1.0.0
Release     : 0.el7
Architecture: noarch
Install Date: (not installed)
Group       : Applications/File
Size        : 9515
License     : GPLv2+
Signature   : RSA/SHA1, Sun 29 Oct 2017 11:59:14 AM MDT, Key ID
c57149538fcc8421
Source RPM  : rpmdbrc-1.0.0-0.el7.src.rpm
Build Date  : Sun 29 Oct 2017 11:52:57 AM MDT
Build Host  : localhost
Relocations : (not relocatable)
Packager    : Minakshi Chavan
Vendor      : Test Project
Summary     : The utility to identify rpm database corruption and
details around it.
Description :
This Package provides the utility to check rpm database
```

```
corruption, fixes it and gives some more details around rpm
database.
```

4.8 Install the rpm.

```
[root@localhost ~]# rpm -ivh
/root/rpmbuild/RPMS/noarch/rpmdbrc-1.0.0-0.el7.noarch.rpm
Preparing...
################################# [100%]
Updating / installing...
   1:rpmdbrc-1.0.0-0.el7
################################# [100%]
Please run rpmdbrc --help to know more about it!
[root@localhost ~]# rpm -q rpmdbrc
rpmdbrc-1.0.0-0.el7.noarch
[root@localhost ~]# rpm -ql rpmdbrc
/usr/sbin/rpmdbrc
/usr/share/man/man1/rpmdbrc.1.gz
[root@localhost ~]# rpm -qd rpmdbrc
/usr/share/man/man1/rpmdbrc.1.gz
[root@localhost ~]# rpm -qi rpmdbrc
Name        : rpmdbrc
Version     : 1.0.0
Release     : 0.el7
Architecture: noarch
Install Date: Sun 29 Oct 2017 12:04:30 PM MDT
Group       : Applications/File
Size        : 9515
License     : GPLv2+
Signature   : RSA/SHA1, Sun 29 Oct 2017 11:59:14 AM MDT, Key ID
c57149538fcc8421
Source RPM  : rpmdbrc-1.0.0-0.el7.src.rpm
Build Date  : Sun 29 Oct 2017 11:52:57 AM MDT
Build Host  : localhost
Relocations : (not relocatable)
Packager    : Minakshi Pushpendra Chavan
Vendor      : Test Project
Summary     : The utility to identify rpm database corruption and
details around it.
```

```
Description :
This Package provides the utility to check rpm database
corruption, fixes it and gives some more details around rpm
database.
```

4.9 Check script execution and man page.

```
[root@localhost ~]# rpmdbrc
USAGE : rpmdbrc [OPTION]
Try --help for more information
[root@localhost ~]# man rpmdbrc
```

FINALLY, Secured the private Key and kept it at some safe location.

```
[root@localhost ~]# gpg --armor --export 8FCC8421 >
minakshi-private-rpm-sign-key.txt
[root@localhost ~]# cat minakshi-private-rpm-sign-key.txt
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2.0.22 (GNU/Linux)

mQENBFn2FRQBCADQl2EiM2wr63MWbLqSZ4giOYzhDd8MLrV6dWbU1AIBZ4RlauvX
/Fa23MHUGmIO4n9JOpA5FdOZmBG9rnsHsF15c102Lp/6s7e2M0JGZMJ4tYL5uVgY
UM0TClwNRFTloPkKzpd9aDwEErZYnuWc/xuCEcOiFTh2e2YYrDGc2CDt8nsWQi5g
uRm34oL2XEcDY1N3tYoLDDdj8BerZYjkA7dGUfKXaImlJI0wmnN379MqPZSuV+Zz
tV8bRyZgGUUhKXYhOZXsg9YvU2bgRhWVhfKL+MQblrj5m/J2zMtCroD7HkwxqT6f
c3RQeX03T3f3do1RBi/9kXQU+k7C7t++5pb5ABEBAAG0MlNuZWhhIEpvc2hpIChS
ZWQgSGF0IEludGVybikgPHNuZWpvc2hpQHJlZGhhdC5jb20+iQE5BBMBAgAjBQJZ
9hUUAhsDBwsJCAcDAgEGFQgCCQoLBBYCAwECHgECF4AACgkQxXFJU4/MhCFkxAf8
D9iGfBwo7xuVOFcMoCJ4hQKgTo2bSXD3tWW15wyeOiUUlHRl1weN4LqHKZeEkUHP
Wn8GPAjHbTI5tgXdh2qYD/R8j4cSYi1C0prJS0xjuroEUDK9YrfR9T2vHmX5/aTU
OwlmgSDAbRHsKSsIDhsucXtk5xs37wC12gqnxamMny9fiBYM7bZJ2zeJRNjy+bQf
1c8Q4FVEWl6EHIHBcmJXZ5Hl7Iwg45QPksRjUbiEWPRttsqGOoo9sQpFpfyUzi00
j9WRBjmfL9HzhykrfiLit+SpVbgBmxbqPmv7Tet9KX3VAGra4wJiqxzRsCjbqKNl
2lLTTlL80iAAKmJU7j/ZobkBDQRZ9hUUAQgApaWjp00Q5/wfQNh3nBtKXKWKFrur
N4LbvymuRQfpcvhPi+c0gq82NMaTlyZ+me/ZCWiPL5+0089rLHwdTmUCMX2+Jb+z
lApnoX1esZu4lgzKgBiz0f7VE9eSkYQLTtLu6SsvQJeQQ5Go1ObUmFkdE10aGgrv
XD+IzKIPSqiujHNpwIFfdsv1Le1QKIHogYzMnK39X35Y7hqCkdtw5Ll0RRz2E2zq
/1ZytgaJCPyU+4f2f0Qpv4x9DpYT08p0Y73VgWadlmkqJnUQeW3MccMw45R/23KW
M3+Jkz9HH1bw77RfUK1Aks+aIusLRnh4ND5+HQxDIxhOU3HHLVdBp/z2PQARAQAB
```

```
iQEfBBgBAgAJBQJZ9hUUAhsMAAoJEMVxSVOPzIQhk3sH/26BrS/yu0kowUnk2qKF
wO2jZ5FCy5KcKSGqHIxcP6Z0nnO1lGdJcq1UtASSypRHwz3Rz8PxFqkq7TNIUDfo
8+JKjWOfBsrHjK3Gjt48Rv6r8QDStWxWPCuZDVgPHyFA+Fn84011CwNYdi3ZA7L1
pVZR8/b2WL5SAjOVho9gMSjNZ4HOllENVbhx2KDu+kWPTaCCG+ix8iro377i1hzp
Nwp8H33YRGcAeOSq+vLCN9bPVAQpzbEhBDSM7XGk5n1fjEhrTvD/sdC7ql0HfCDK
fT8gLo4oy4JlFfZz353rJBPRX0RtgiWG9tuhj0E2qGxmLg5O0KWey9M/kgMhFELD
S2E=
=+Xyd
-----END PGP PUBLIC KEY BLOCK-----
```

Chapter 5
Future Scope & Conclusion

5.1 Package Delivery Mechanism: There are multiple package delivery mechanisms, one of which is discussed in this project i.e. rpm based deployments. The most famous one is the OTA update. An over-the-air (OTA) update is the wireless delivery of new software, firmware, or other data to mobile devices.

Wireless carriers and original equipment manufacturers (OEMs) typically use over-the-air updates to deploy firmware and configure phones for use on their networks over Wi-Fi or mobile broadband. The initialization of a newly purchased phone, for example, requires an over-the-air update. With the rise of smartphones, tablets and internet of things (IoT) devices, carriers and manufacturers have turned to different over-the-air update architecture methods for deploying new operating systems (OSes) to these devices.

The rpm os-tree is a new method. rpm-ostree is a hybrid image/package system. It uses libOSTree as a base image format, and accepts RPM on both the client and server side, sharing code with the dnf project; specifically libdnf.

The OSTree related projects section covers this to a degree. As soon as one starts taking "snapshots" or keeping track of multiple roots, it uncovers many issues. For example, which content specifically is rolled forward or backwards? If the package manager isn't deeply aware of a snapshot tool, it's easy to lose coherency. A concrete example is that rpm-ostree moves the RPM database to /usr/share/rpm, since we want one per root /usr. In contrast, the snapper tool goes to some effort to include /var/lib/rpm in snapshots, but avoids rolling forward/back log files in /var/log. OSTree requires clear rules around the semantics of directories like /usr and /var across upgrades, and while this requires changing some software, we believe the result is significantly more reliable than having two separate systems like yum and snapper glued together, or apt-get and BTRFS, etc.

Furthermore, beyond just the mechanics of things like the filesystem layout, the implemented upgrade model affects the entire user experience. For example, the base system OSTree commits that one replication from a remote server can be assigned version numbers. They are released as coherent wholes, tested together. If one is simply performing snapshots on the client side, every client machine can have different versions of components. Related to this is that rpm-ostree clearly distinguishes which packages you have layered, and it's easy to remove them, going back to a pristine, known state. Many package managers just implement a "bag of packages" model with no clear bases or layering. As the OS evolves over time, "package drift" occurs where you might have old, unused packages lying around.

5.2 Benefits of RPM Method

RPMs not only make it easy for the user to install software on their computer but also for the developer to deliver the software. RPMs make it easy to pull in dependencies, other bits of code needed by the software to function properly, and to provide updates to the software in question. The ability to apply patches for security fixes makes RPMs an especially good tool for maintaining secure computer environments as code fixes can easily be verified by system administrators prior to installation.

Package repositories can also be made to allow users access to a central database of software that is easily installed. The user can determine where the software originated and once installed is prompted to perform any upgrades when updates are available in the repository. The user can remove the software at any time and the RPM installer will automatically clean up the installation, preventing old versions of the software from persisting on the system, which could get used by mistake or expose the user to flaws or exploits. Removing unused software reduces potential attack vectors and because RPMs make it easy to remove unused software, users are much more likely to do so.

Chapter 6
CONCLUSION

It is possible to deliver any payload in an Operating system using RPM based deployment which not only provides authenticity, integrity and validations. The project demonstrates a very easy way to not only build an rpm but also deploy it with full integrity and security features not only to utilize the contents but also to verify those all.

Chapter 7
REFERENCES

[1] Red Hat Documentations
https://www.redhat.com/en/blog/security-benefits-rpm-packaging

[2] Open Source Articles
https://opensource.com/article/18/7/evolution-package-managers

[3] Fedora Documentation
https://rpm-packaging-guide.github.io/

[4] A management system for software package distribution
https://ieeexplore.ieee.org/abstract/document/6304372
By - Sophon Mongkolluksame; Chavee Issariyapat; Panita Pongpaibool; Koonlachat Meesublak; Nontaluck Nulong; Sirikarn Pukkawanna
Date of Conference: 29 July 2012 - 02 August 2012
Date Added to IEEE Xplore: 17 September 2012
ISBN Information:
Print ISSN: 2159-5100
INSPEC Accession Number: 12999654
Publisher: IEEE
Conference Location: Vancouver, BC, Canada

[5] A graph method of package dependency analysis on Linux Operating system
https://ieeexplore.ieee.org/abstract/document/7490780
By - Jing Wang; Qingbo Wu; Yusong Tan; Jing Xu; Xiaoli Sun
Published in: 2015 4th International Conference on Computer Science and Network Technology (ICCSNT)
Date of Conference: 19-20 December 2015
Date Added to IEEE Xplore: 16 June 2016
ISBN Information:
Electronic ISBN:978-1-4673-8173-4
CD:978-1-4673-8172-7
INSPEC Accession Number: 16090264
DOI: 10.1109/ICCSNT.2015.7490780
Publisher: IEEE
Conference Location: Harbin

[6] Package Management System in Linux
https://ieeexplore.ieee.org/abstract/document/9544805

By - Shrinidhi G Hegde; G Ranjani
Published in: 2021 Asian Conference on Innovation in Technology (ASIANCON)
Date of Conference: 27-29 August 2021
Date Added to IEEE Xplore: 04 October 2021
ISBN Information:
Electronic ISBN:978-1-7281-8402-9
CD:978-1-7281-8400-5
USB ISBN:978-1-7281-8401-2
Print on Demand(PoD) ISBN:978-1-7281-8403-6
INSPEC Accession Number: 21202856
DOI: 10.1109/ASIANCON51346.2021.9544805
Publisher: IEEE
Conference Location: PUNE, India

[7] Understanding and Auditing the Licensing of Open Source Software Distributions
https://ieeexplore.ieee.org/abstract/document/5521758
By - Daniel M. German; Massimiliano Di Penta; Julius Davies
Published in: 2010 IEEE 18th International Conference on Program Comprehension
Date of Conference: 30 June 2010 - 02 July 2010
Date Added to IEEE Xplore: 26 July 2010
ISBN Information:
Electronic ISBN:978-1-4244-7603-9
Print ISBN:978-1-4244-7604-6
Online ISBN:978-0-7695-4113-6
ISSN Information:
Print ISSN: 1092-8138
Electronic ISSN: 1092-8138
INSPEC Accession Number: 11453730
DOI: 10.1109/ICPC.2010.48
Publisher: IEEE
Conference Location: Braga, Portugal