



# “Advanced Technique and Solution to Manage Linux Packages with Lua”

**Rahul Dalshringar Yadav<sup>1</sup>**

Student, Rizvi college of Engineering (Mumbai, India), yadavrahul@eng.rizvi.edu.in

**Aadarshkumar Hakimchand Sharma<sup>2</sup>**

Student, Rizvi college Engineering (Mumbai, India), aadarsh123@eng.rizvi.edu.in

**Mohanish Madanlal Rajak<sup>3</sup>**

Student, Rizvi college of Engineering (Mumbai, India), mohanish@eng.rizvi.edu.in

**Sahil Valanju<sup>4</sup>**

Student, Rizvi college of Engineering (Mumbai, India), sahilvalanju@eng.rizvi.edu.in

**Anupam Choudhary<sup>5</sup>**

Professor Rizvi college of Engineering (Mumbai, India), anupamchoudhary@eng.rizvi.edu.in

## Abstract

This paper is a Lua-based application proposed to function as a package manager for Linux. This application is not just a tool, but a comprehensive solution for managing software packages on a Linux system. It simplifies the process of software management, here by enhancing the user experience.

This proposed primary function of the application is to allow users to download and install applications from a package database using simple commands. This feature eliminates the need for users to manually search for software online, download them, and then go through the often-complicated process of installation. Instead, all these steps are consolidated into one simple command, making the process much more efficient and user-friendly.

In addition to installation, the application will provide functionalities for searching the package list. This means that users can easily find out what software packages are available for installation. This is particularly useful when users are not sure about the name of a specific package or when they want to explore what other software options area valuable.

The application will also allow users to retrieve package information. This feature provides users with details about a specific package, such as its version, size, dependencies, and more. This information can be crucial when deciding whether to install a particular package or when troubleshooting issues related to software installation.

Another key feature of this application will be its ability to delete packages. Just like the installation process, uninstalling software can also be complicated and time-consuming. However, with this application, users can easily uninstall any unwanted software with a simple command.

The use of Lua, a powerful, efficient, lightweight, embeddable scripting language, ensures that the application is robust and versatile. Lua's efficiency means that the application runs smoothly without consuming excessive system resources. Its versatility allows it to handle various software management tasks with ease.

The research will aim to streamline the process of software installation and management on Linux systems. By consolidating multiple tasks into one application and simplifying the user interface, it makes software management more accessible and user-friendly. Whether you're a seasoned Linux user or a beginner just starting out, this Lua-based package manager can make your Linux experience much more enjoyable and efficient.

In conclusion, this paper represents a significant advancement in Linux software management. By leveraging the power of Lua and focusing on user experience, it has the potential to revolutionize how we install, manage, and uninstall software on Linux systems.

**Keywords:** Lua, Package manage and Linux, multiparadigm ,API

## 1. Introduction

This research is not just a tool, but a platform that you can use to implement in your own projects. You have the freedom to customize it, even change its name, to suit your specific needs. In the vast landscape of Linux package management, our program stands out as a unique and powerful framework. It is designed with flexibility and customization in mind, allowing users to tailor the program to their specific needs. Whether you're a seasoned developer or a beginner just starting out, our program offers the tools and features you need to manage software packages on Linux systems effectively. One of the key features of our program is its ability to allow users to implement it in their own projects.

This means that you can take our framework and build upon it, adding your own features and functionalities. This level of customization is rarely seen in other package managers and sets our program apart. Furthermore, our research allows users to change its name. This might seem like a small feature, but it can be crucial when integrating the program into your own projects. By allowing you to rename the program, we ensure that it seamlessly fits into your project's ecosystem. In our investigation of existing package managers, we found that they typically include only their own databases. This can be limiting as it restricts users to the software packages available in those databases. Our scheme, however, offers a unique feature: the ability for users to add their own databases.

This flexibility allows for a more personalized and efficient package management experience. The ability to add your own database means that you can curate a list of software packages that are relevant to your specific needs. Whether you're working on a personal project or managing software for a large organization, this feature allows you to create a customized software repository. While other package managers have features called mirrors, which are essentially copies of software repositories hosted in different locations, our package manager takes it a step further. It allows users to change the entire database if they need to.

This level of customization is unprecedented and opens up new possibilities for package management on Linux systems. Imagine being able to swap out an entire database of software packages with another one that better suits your needs. With our technique, this is not just possible but easy and straightforward. Whether you want to switch to a database with more up-to-date packages or one that includes software tailored to a specific industry, our scheme makes it possible. In conclusion, our program is more than just a package manager.

It's a flexible framework that puts the power of customization in your hands. Whether you're looking to implement it in your project or modify it for your personal use, our research offers the tools and flexibility you need. By providing a customizable framework for Linux package management, we aim to empower users and developers alike. We believe that by giving you the tools and freedom to customize your package management experience, we can help foster innovation and efficiency in the Linux community.

## 2: Review of Literature

2.1. In this literature explores the intricate world of GNU/Linux distributions and package management, drawing analogies to the differences between small village and large cities.

In a small village, everyone knows each other, conflicts are visible, and order is maintained without a formal bureaucracy. In contrast, a large city is characterized by anonymity, lack of personal connections, and the need for bureaucratic structures to maintain order.

The author likens GNU/Linux distributions to a city due to their complexity. A GNU/Linux distribution comprises thousands of software packages, making it impossible to have a functional distribution with less than a few hundred packages. Package management serves as the "bureaucratic order" that tracks and organizes all these software pieces to maintain civil order and avoid conflicts.

Distinguishing GNU/Linux distributions from proprietary operating systems like Windows and macOS, the thesis emphasizes the user's greater control and flexibility in shaping the operating system. This flexibility can address diverse needs and preferences, from bug fixes and security upgrades to customization for specific hardware.

The literature is structured into four sections. The first delves into optimizing a GNU/Linux system, covering aspects such as software code optimization, file systems, and kernels. Benchmarking techniques are explored in detail. This provides insights into the internal workings of GNU/Linux systems and package management, from both developer and user perspectives.

This delves into the requirements for creating a package manager, emphasizing practical implementation and the strengths and weaknesses of existing package management systems. It discusses how developers can maintain a package manager, focusing on the broader picture of a GNU/Linux distribution's role in the real world.

The literature begins by defining a GNU/Linux distribution as an operating system consisting of the Linux kernel and the GNU user space. These distributions organize and distribute software licensed under the GPL, with over three hundred and fifty active distributions offering various advantages.

The primary distinctions among distributions are based on their orientation toward specific user types. Some cater to novices, offering user-friendly GUI installation procedures, while others target advanced users, providing more configuration options and documentation in the form of "How-tos." Package management systems, hardware support, and software optimization vary accordingly.

The literature acknowledges the challenges facing GNU/Linux distributions, including optimizing software packages for different processors, balancing stability with up-to-date software, and simplifying usability for both novice and advanced users. It emphasizes the importance of maintaining consistency in configuration styles and addressing the balance between configurability and user-friendliness.

Review provides a comprehensive examination of GNU/Linux distributions, package management, and the intricacies of managing software in this complex ecosystem, drawing parallels between software management and the dynamics of small villages and larger cities.

The text delves into the concept of optimization in the context of GNU/Linux distribution development. The chapter emphasizes that optimization aims to make something perform at its best, with a particular focus on speed. It also underscores that stability is a crucial aspect of optimization, as an unstable system cannot be considered optimized. It discusses the choices available to developers of GNU/Linux distributions in terms of optimizing their systems. It points out that stability optimizations often stem from policies enforced by the package management system, while speed optimizations come from the developer's experience in configuring the system with various patches, options, and packages.



Additionally, the text introduces the concept of optimizing the Linux kernel for speed and describes the various branches of the Linux kernel, such as vanilla-sources, mm-sources, and ck-sources. Each of these branches is associated with different features and optimizations, with a focus on CPU schedulers and memory management features.

The text also explains strategies for optimizing the kernel to reduce memory consumption, including the use of modules and initramfs images. These methods allow the kernel to be tailored to specific hardware configurations without wasting memory resources.[1]

## 2.2.pacman – ArchWiki

In this official Arch Wiki of pacman, suggests Pacman to be a major distinguishing feature of Arch Linux. Pacman is the official package manager of Arch Linux, pacman keeps the system up-to-date by synchronizing package lists with the master server. With the help of this particular distinguishing ability we understood that a list is kept up-to-date which is responsible to list most recent update packages and their downloads and this package list is present in a database. This type of model is known as the Server and client model.

A package is an archive containing: (Arch wiki)

- all of the (compiled) files of an application
- metadata about the application, such as application name, version, dependencies, etc.
- installation files and directives for *pacman*
- (optionally) extra files to make your life easier, such as a start/stop script

Some packages require additional dependencies. Packages themselves file this dependency with respect to the package manager. This particular wiki article is essential for understanding our package manager because the use of this type of package manager is popular for Linux-based distros, pacman is made in C and uses the *bsdtar* format. Packaging this sets understanding of actual formatting and packaging packages in the dependencies box.

In this research, suggest how Linux's software packaging has changed over time. Initially, tar balls were used to package software and send it to various systems for installation. A tar ball, which may be constructed with just one tar command and typically contains numerous files, is a single archive file. However, there were a number of drawbacks to this approach, including difficulties in administering the software after installation, an inability to determine the software version, and difficulties in upgrading the software. Additionally, the user had to manually install any dependencies that the product had on other programs. Due to these difficulties, package-management systems, or package managers, have become widely used. By packaging meta-data along with the actual software, package managers aid in the resolution of software management and version number issues. A complete description of the number of files in the package, a list of all files, the version number, a description of the package, information about the packager, and any other information the developer may wish to include are all included in the meta-data. The meta-data may also contain scripts that convey commands to do operations like creating directories, adding user accounts, or turning on services, as well as dependencies required for the software packages to function. An in-depth analysis of these systems and their historical development is the goal of this paper.[4]

Table 1: Summary of package managers

Package manager	Binary/Source	Install from URL support	GUI enabled	File Extension
APT	Binary	No	Yes	.deb
DNF	Binary	Yes	Yes	.rpm
Pacman	Binary	Yes	No	.pkg.tar.xz
Portage	Source	No	No	.ebuild
YUM	Binary	Yes	Yes	.rpm
ZYpp	Binary	Yes	Yes	.rpm

### 2.3. Kaspersky Security Bulletin 2021 Statistics

Kaspersky lab is a reputed company that deals with cybersecurity threats and solutions, according to Kaspersky Security Bulletin 2021 Statistics, Kaspersky Lab divides various types of attack vectors such as financial threats, Ransomware program, Miners, etc in parts and represent statistics in the form of graph pie chart and brief. : According to Kaspersky Lab's security bulletin statistics 2021 more than 114,525,734 unique malicious URLs triggered web Anti-virus components, most of this malwares were caught because of clicking malicious links which were pasted on unverified software solution .[3]

### 2.4. Graphical representation of Lua versioning

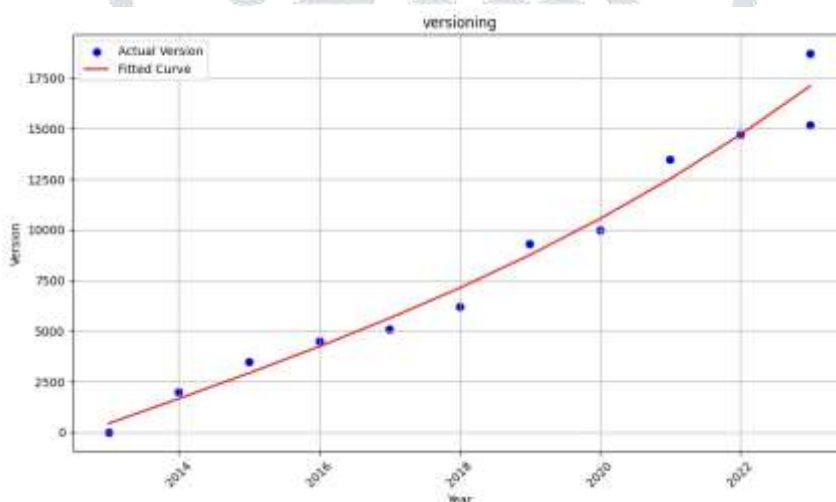


Fig. no.1. Version of Lua based on years

As shown in Fig.no.1, the initial version of the lua started in 2013 as time goes increase the versioning of the lua goes on the increasing and it reach to the 18k. the latest version of the lua is 5.4.1.

## 3. Process of Designing:

### 3.1. Outline

This research adopts a user-centric approach, providing a malleable framework that can be modified to align with individual requirements.

1. It encourages users to incorporate it into their own projects, enhancing its functionality with their unique features, and even rebranding it if necessary.
2. A distinguishing feature of this methodology is the provision for users to supplement their own databases, thereby personalizing their package management experience.
3. The scheme also pioneers a novel concept in package management by enabling users to replace the entire database, thereby offering an unprecedented level of customization. This allows users to switch to a database that better aligns with their needs, whether it be one with more current packages or one tailored to a specific industry.

4. The underlying principle of the research is the belief in user empowerment through customization. This is based on the idea that adaptability leads to increased efficiency and fosters innovation.
5. The scheme adds up to the theory that a tool’s effectiveness is amplified when it can be tailored to the user’s specific needs, thereby enhancing their package management experience.

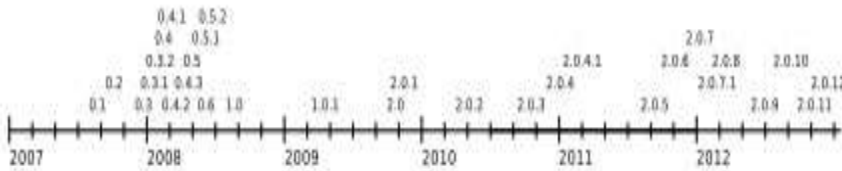
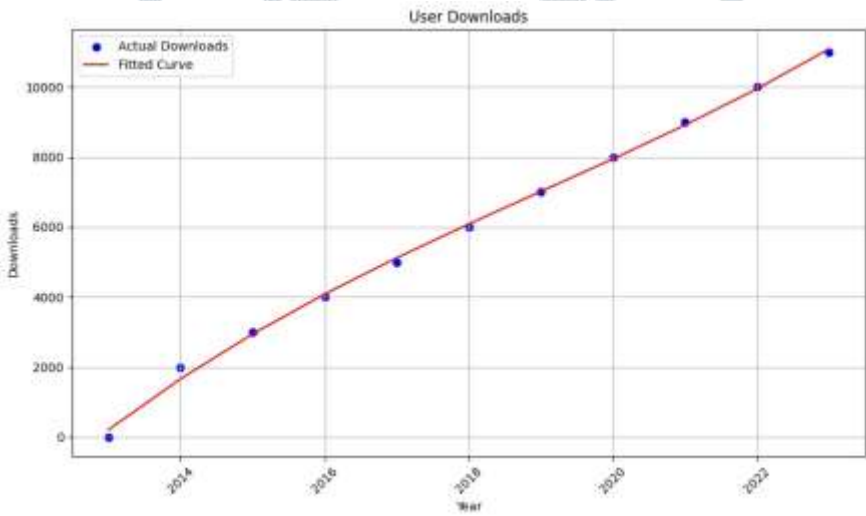


Fig. No.2 . Timeline of LuaRocks releases

In the above fig as we can see the most of the lua version were created in the 2008 which was the lua came into the picture of the package manager. For the certain period of time the updating of the luarock is not much more but as we get introduce more about the luarock there are some updated versions came into the picture with the various modules and packages. the latest version 5.4 till the 2023, there are thousands of the version of the luarocks are created till now with the updating of its feature and modules.



Graph No.1: Downloading of lua software by user over the years.

The line graph shows the relationship between user downloads and filter curve. The x-axis represents the year, and the y-axis represents the number of user downloads. The blue line represents the actual number of downloads, and the red line represents the fitted curve.

The graph shows that the number of user downloads has increased steadily over time. The fitted curve shows that the growth in user downloads is exponential. This suggests that the filter curve is becoming increasingly popular with users.

There are a few possible explanations for this trend. One possibility is that the filter curve is becoming more effective at filtering out unwanted content. This would make it more attractive to users who are looking for a more curated experience.

Another possibility is that the filter curve is becoming more customizable. This would allow users to tailor the filter curve to their individual needs and preferences. This would also make it more attractive to a wider range of users.

Overall, the graph suggests that the filter curve is becoming increasingly popular with users. This is likely due to a combination of factors, including its effectiveness, customizability, and increasing popularity with content creators.

Here are some additional insights that can be drawn from the graph:

- The growth in user downloads has been particularly strong in recent years.
- The fitted curve suggests that the growth in user downloads is likely to continue in the future.
- The filter curve is more popular in some years than others. For example, there was a surge in downloads in 2020. This may be due to the COVID-19 pandemic, which led to more people spending time online.

Overall, the graph provides a valuable overview of the popularity of the filter curve over time. It is clear that the filter curve is becoming increasingly popular with users, and this trend is likely to continue in the future.

### 3.2. Methodology

1. Database Creation: The database is the heart of your package manager. It should be designed to efficiently store and retrieve information about software packages. This includes the package name, version, dependencies, download link, and installation instructions. Depending on your needs, you might want to consider using a structured format like SQL or a NoSQL database. The database should be designed with performance in mind to ensure quick retrieval of package information.
2. User Interface: The user interface should be intuitive and easy to use. Since you're designing for a Linux environment, a command-line interface (CLI) would be most appropriate. The CLI should support various commands like install, update, delete, list, and about. Each command should have clear and concise documentation that users can access with a `--help` flag. The user interface should also handle errors gracefully and provide useful feedback to the user.
3. Customization: Allowing users to customize the database adds flexibility to your package manager. Users can choose to add or remove servers from which packages are fetched. This could be implemented by allowing users to add or remove entries in a configuration file.

### 3.3: Algorithm

Step 1: Define a function `install(package)` that takes a package name as an input and installs the package on the system.

Step 2: Inside the function, check if the package is already installed on the system. If yes, return "Package already installed".

Step 3: If not, check if the package is available in the online repository. If no, return "Package not found".

Step 4: If yes, check if the package has any dependencies. If yes, call the function `install(dependency)` for each dependency.

Step 5: Download the package from the online repository and verify its integrity.

- Step 6: Extract the package files and copy them to the appropriate locations on the system.
- Step 7: Update the database of installed packages and their versions.
- Step 8: Return "Package installed successfully".
- Step 9: Repeat steps 1 to 8 for the functions ``update(package)``, ``remove(package)``, ``list()``, and ``search(query)``, with appropriate modifications according to their specifications.

3.4:Execution process of Lua:

3.4.1:Flow chart:

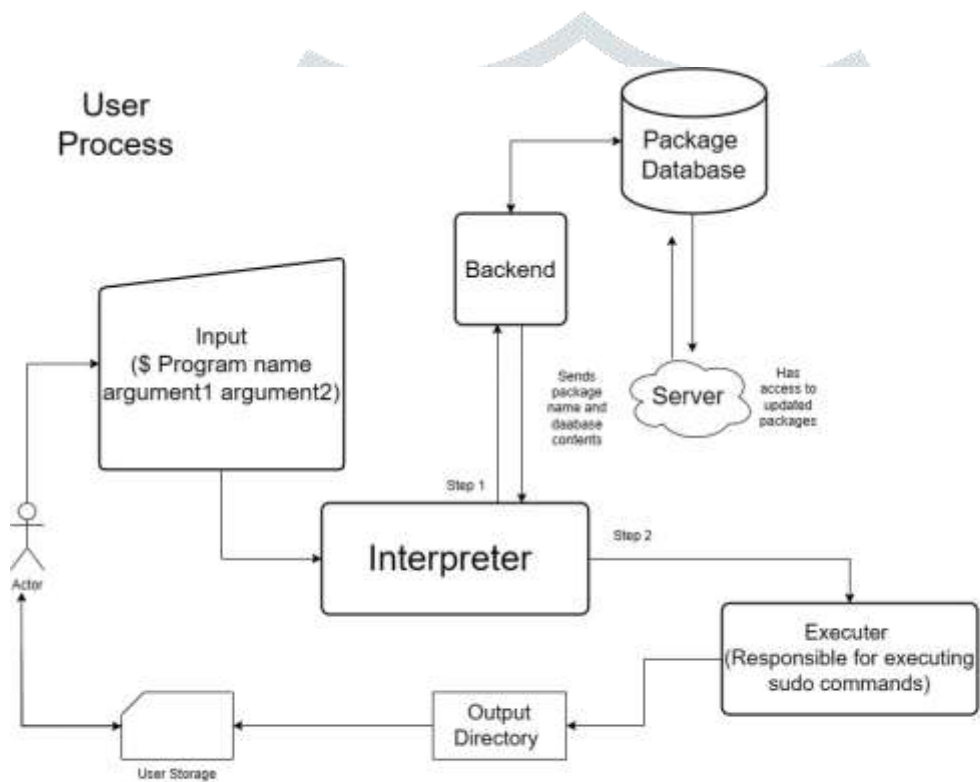


Fig.No.3: user interaction with databases and Lua interpreter



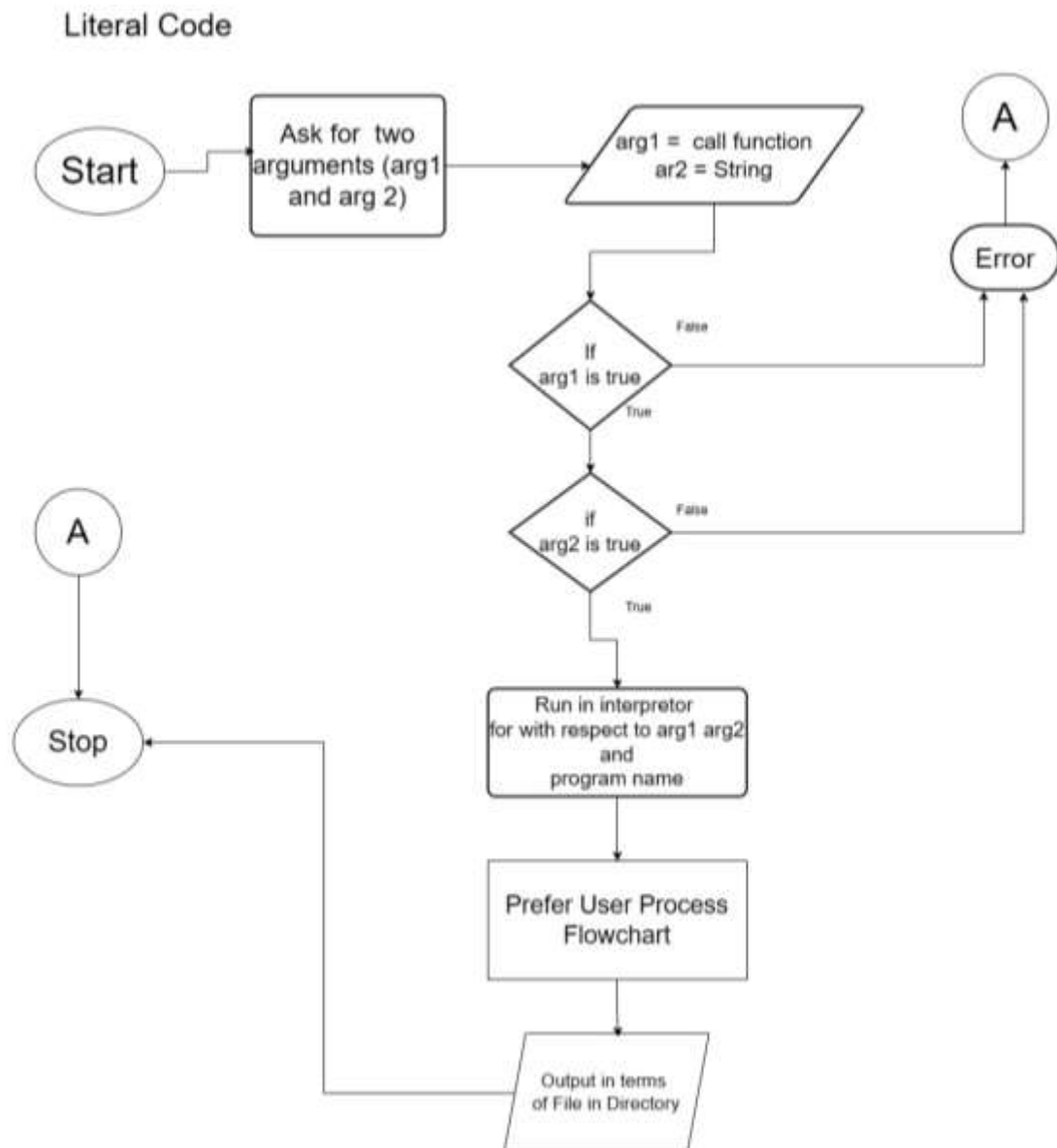


Fig No.4: Execution process of Lua

The two flowcharts show different parts of the same process: a user process for interacting with a package database.

The first flowchart shows the overall process, from the user inputting the program name and arguments to the package being installed in the output directory.

The second flowchart shows the details of the "If" decision block in the first flowchart. It shows how the interpreter determines which package database to use and how it sends the package name and database contents to the server.

Here is a combined description of the two flowcharts:

Start

1. User inputs the program name and any necessary arguments.
2. The interpreter parses the input and determines which package database to use. This is done by checking the program name against a list of known package databases.

3. If the program name is not found in the list of known package databases, the interpreter returns an error.
4. Otherwise, the interpreter sends the package name and database contents to the server.
5. The server has access to updated packages, and it sends the requested package to the executor.
6. The executor is responsible for executing sudo commands, which allows the user to install the package.
7. The package is installed in the output directory.
8. The user can then access the package from their user storage.

Stop

The flowchart also shows a "Prefer User Process" block. This block indicates that the user process should be used if it is available. This is because the user process may be more efficient than the standard package database process.

Here is a more detailed description of the "If" decision block:

If arg1 is true

- If arg2 is true
- Run in interpreter for with respect to arg1 arg2 and program name
- Else
- Output in terms of File in Directory

The "arg1" and "arg2" variables are not defined in the flowchart, so it is difficult to say exactly what they represent. However, it is possible that they are used to specify the package database and the directory to install the package in, respectively.

The "Run in interpreter for with respect to arg1 arg2 and program name" block indicates that the interpreter should be used to install the package. This may be necessary if the package is not available in the standard package database.

The "Output in terms of File in Directory" block indicates that the output of the process should be written to a file in the specified directory. This may be necessary if the package is being installed for a specific user or if the installation process needs to be logged.

Overall, the two flowcharts provide a comprehensive overview of the user process for interacting with a package database. The combined description above provides a more detailed explanation of the steps involved in the process.

### 3.5:Process of installation:

#### 3.5.1: Lua installation process

As shown in successive outputs Lua installations along with commands and related outputs has been executed and our installation process is fast and flexible that supports the user to do as per his requirements

```
~/code via 2 v3.10.4 took 5s
$ sudo apt install lua5.4
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  brave-keyring cpu-checker dmccvntd gir1.2-gtk-vnc-2.0 gir1.2-libosinfo-1.0 gir1.2-spiceclientglib-2.0 gir1.2-spiceclientgtk-3.0
  ibverbs-providers ipxe-qemu ipxe-qemu-256k-compat-efi-roms jq libaio1 libcacard0 libdaxctl1 libdevmapper-event1.02.1 libfdt1
  libgfat0 libgfrpc0 libgfxdr0 libglusterfs0 libgovirt-common libgovirt2 libgtk-vnc-2.0-0 libgvnc-1.0-0 libhverbai libiscsi7 libjql
  liblvm2cmd2.03 libndctl6 libonig5 libosinfo-1.0-0 libphodav-2.0-0 libphodav-2.0-common libpmem1 libpmemobj1 libqrencode4 librados2
  librbd1 librdmac1 libspice-client-glib-2.0-0 libspice-client-gtk-3.0-5 libspice-server1 libtpms0 liburing2 libusbredirhost1
  libusbredirparser1 libvirglrenderer1 lvm2 mdevctl msr-tools osinfo-db ovmf pass python3-libxml2 qrencode seabios
  spice-client-glib-usb-acl-helper swtpm swtpm-tools thin-provisioning-tools uidmap
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed:
  lua5.4
0 upgraded, 1 newly installed, 0 to remove and 6 not upgraded.
1 not fully installed or removed.
Need to get 163 kB/177 kB of archives.
After this operation, 464 kB of additional disk space will be used.
Get:1 http://in.archive.ubuntu.com/ubuntu jammy/universe amd64 lua5.4 amd64 5.4.4-1 [163 kB]
Fetched 163 kB in 1s (323 kB/s)
Selecting previously unselected package lua5.4.
(Reading database ... 297660 files and directories currently installed.)
Preparing to unpack .../lua5.4_5.4.4-1_amd64.deb ...
Unpacking lua5.4 (5.4.4-1) ...
Setting up lua5.4 (5.4.4-1) ...
update-alternatives: using /usr/bin/lua5.4 to provide /usr/bin/lua (lua-interpreter) in auto mode
update-alternatives: using /usr/bin/luac5.4 to provide /usr/bin/luac (lua-compiler) in auto mode
```

#### Installing Lua on Linux

Lua is a powerful, lightweight, embeddable scripting language with a wide range of applications. It is a popular choice for game development, web applications, and embedded systems. There are two primary methods for installing Lua on Linux: using the package manager or compiling from source.

#### Method 1: Using the Package Manager

The simplest and most recommended method for installing Lua is to use the package manager provided by your Linux distribution. This method ensures that you get the latest stable version of Lua for your system and that all of the necessary dependencies are installed automatically.

Here are the steps on how to install Lua using the package manager:

1. Open a terminal window.
2. Update the package manager's index:

#### Bash

```
sudo apt update (Debian-based distributions)
sudo dnf update (Fedora-based distributions)
sudo pacman -Syu (Arch Linux)
```

3. Install the Lua package:

#### Bash

```
sudo apt install lua5.4 (Debian-based distributions)
sudo dnf install lua (Fedora-based distributions)
sudo pacman -S lua (Arch Linux)
```

## Method 2: Compiling from Source

Compiling Lua from source is a more complex process, but it gives you more control over the installation and allows you to install specific versions of Lua or modify the build process.

Here are the steps on how to compile Lua from source:

1.Download the latest stable version of Lua from the official

2.Extract the downloaded tarball:

```
tar xzf lua-5.4.6.tar.gz
```

3.Change directory to the extracted Lua source directory:

```
cd lua-5.4.6
```

4.Configure Lua for your system:

```
./configure
```

5.Build Lua:

```
make
```

6.Install Lua:

```
sudo make install
```

A terminal window with a dark background. The prompt is ~/code. The user enters 'lua' and the terminal shows 'Lua 5.4.4 Copyright (C) 1994-2022 Lua.org, PUC-Rio' followed by a cursor.


```
~/code via 🌙 v5.4.4 via 🐉 v3.10.4  
> lua  
Lua 5.4.4 Copyright (C) 1994-2022 Lua.org, PUC-Rio  
> █
```

## Testing the Lua Installation

Once Lua is installed, you can test your installation by running the following command in a terminal window:

```
lua -v
```

This command should print the version of Lua that you installed.

A terminal window with a dark background. The prompt is ~/code. The user enters 'lua -v' and the terminal shows 'Lua 5.4.4 Copyright (C) 1994-2022 Lua.org, PUC-Rio' followed by a cursor.

```
~/code via 🌙 v5.4.4 via 🐉 v3.10.4  
> lua -v  
Lua 5.4.4 Copyright (C) 1994-2022 Lua.org, PUC-Rio  
~/code via 🌙 v5.4.4 via 🐉 v3.10.4  
> █
```



Version checking and correction

```
~/code via 🌙 v5.4.4
> wget http://www.lua.org/ftp/lua-5.4.4.tar.gz
--2022-07-30 11:52:06-- http://www.lua.org/ftp/lua-5.4.4.tar.gz
Resolving www.lua.org (www.lua.org)... 88.99.213.221, 2a01:4f8:10a:3edc::2
Connecting to www.lua.org (www.lua.org)|88.99.213.221|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 360876 (352K) [application/gzip]
Saving to: 'lua-5.4.4.tar.gz'

lua-5.4.4.tar.gz      100%[=====] 352.42K  44.8KB/s   in 7.6s
2022-07-30 11:52:14 (46.2 KB/s) - 'lua-5.4.4.tar.gz' saved [360876/360876]

~/code via 🌙 v5.4.4 took 8s
> 
```

Some packages of Lua has been installed

### 3.5.2:Packages and related

This show the all the packages available in the lua rock package manager.in the below output we mentioned all the package information.

```
~/code via 🌙 v5.4.4 took 8s
> tar xvfz lua-5.4.4.tar.gz
lua-5.4.4/
lua-5.4.4/Makefile
lua-5.4.4/doc/
lua-5.4.4/doc/luac.1
lua-5.4.4/doc/manual.html
lua-5.4.4/doc/manual.css
lua-5.4.4/doc/contents.html
lua-5.4.4/doc/lua.css
lua-5.4.4/doc/osi-certified-72x60.png
lua-5.4.4/doc/logo.gif
lua-5.4.4/doc/lua.1
lua-5.4.4/doc/index.css
lua-5.4.4/doc/readme.html
lua-5.4.4/src/
lua-5.4.4/src/ldblib.c
lua-5.4.4/src/lmathlib.c
lua-5.4.4/src/loslib.c
```

Changed the directory in Lua: this show the way of changing the directory in the lua.

```
~/code via 🌙 v5.4.4
> cd lua-5.4.4/

~/code/lua-5.4.4
> make linux install INSTALL_TOP=/usr/local/lua/5.4.4 MYLIBS="-lncurses"
make[1]: Entering directory '/home/meet/code/lua-5.4.4/src'
make all SYSCFLAGS="-DLUA_USE_LINUX" SYSLIBS="-Wl,-E -ldl"
make[2]: Entering directory '/home/meet/code/lua-5.4.4/src'
gcc -std=gnu99 -O2 -Wall -Wextra -DLUA_COMPAT_5_3 -DLUA_USE_LINUX -c -o ldblib.o ldblib.c
gcc -std=gnu99 -O2 -Wall -Wextra -DLUA_COMPAT_5_3 -DLUA_USE_LINUX -c -o liolib.o liolib.c
gcc -std=gnu99 -O2 -Wall -Wextra -DLUA_COMPAT_5_3 -DLUA_USE_LINUX -c -o lmathlib.o lmathlib.c
gcc -std=gnu99 -O2 -Wall -Wextra -DLUA_COMPAT_5_3 -DLUA_USE_LINUX -c -o loadlib.o loadlib.c
gcc -std=gnu99 -O2 -Wall -Wextra -DLUA_COMPAT_5_3 -DLUA_USE_LINUX -c -o loslib.o loslib.c
gcc -std=gnu99 -O2 -Wall -Wextra -DLUA_COMPAT_5_3 -DLUA_USE_LINUX -c -o lstrlib.o lstrlib.c
gcc -std=gnu99 -O2 -Wall -Wextra -DLUA_COMPAT_5_3 -DLUA_USE_LINUX -c -o ltablib.o ltablib.c
gcc -std=gnu99 -O2 -Wall -Wextra -DLUA_COMPAT_5_3 -DLUA_USE_LINUX -c -o lutf8lib.o lutf8lib.c
gcc -std=gnu99 -O2 -Wall -Wextra -DLUA_COMPAT_5_3 -DLUA_USE_LINUX -c -o linit.o linit.c
ar rcu liblua.a capi.o lcode.o lctype.o ldebug.o ldo.o ldump.o lfunc.o lgc.o llex.o lmem.o lobject.o lopcodes.o lparser.o
.o lzio.o lauxlib.o lbaselib.o lcorolib.o ldblib.o liolib.o lmathlib.o loadlib.o loslib.o lstrlib.o ltablib.o lutf8lib.o
ar: 'u' modifier ignored since 'D' is the default (see 'U')
ranlib liblua.a
gcc -std=gnu99 -O2 -Wall -Wextra -DLUA_COMPAT_5_3 -DLUA_USE_LINUX -c -o lua.o lua.c
gcc -std=gnu99 -o lua lua.o liblua.a -lm -Wl,-E -ldl -lncurses
```

```
~/code/lua-5.4.4 took 4s
> make linux test
make[1]: Entering directory '/home/meet/code/lua-5.4.4/src'
make all SYSCFLAGS="-DLUA_USE_LINUX" SYSLIBS="-Wl,-E -ldl"
make[2]: Entering directory '/home/meet/code/lua-5.4.4/src'
gcc -std=gnu99 -o lua lua.o liblua.a -lm -Wl,-E -ldl
gcc -std=gnu99 -O2 -Wall -Wextra -DLUA_COMPAT_5_3 -DLUA_USE_LINUX -c -o luac.o luac.c
gcc -std=gnu99 -o luac luac.o liblua.a -lm -Wl,-E -ldl
make[2]: Leaving directory '/home/meet/code/lua-5.4.4/src'
make[1]: Leaving directory '/home/meet/code/lua-5.4.4/src'
make[1]: Entering directory '/home/meet/code/lua-5.4.4/src'
./lua -v
Lua 5.4.4 Copyright (C) 1994-2022 Lua.org, PUC-Rio
make[1]: Leaving directory '/home/meet/code/lua-5.4.4/src'

~/code/lua-5.4.4
> sudo make install
[sudo] password for meet:
cd src && mkdir -p /usr/local/bin /usr/local/include /usr/local/lib /usr/local/man/man1 /usr/l
cd src && install -p -m 0755 lua luac /usr/local/bin
cd src && install -p -m 0644 lua.h luaconf.h lualib.h lauxlib.h lua.hpp /usr/local/include
cd src && install -p -m 0644 liblua.a /usr/local/lib
cd doc && install -p -m 0644 lua.1 luac.1 /usr/local/man/man1

~/code/lua-5.4.4 took 3s
> lua
Lua 5.4.4 Copyright (C) 1994-2022 Lua.org, PUC-Rio
> □
```

Linux testing is done: It test the correction of the lua installation in the system and run some of the basic command on it.

```
~/code via 🌙 v5.4.4 via 🐙 v3.10.4 took 2m37s  
> lua  
Lua 5.4.4 Copyright (C) 1994-2022 Lua.org, PUC-Rio  
> 4*8  
32  
> 40//3  
13  
> 40/3  
13.333333333333  
> true and false  
false  
> true and true  
true  
> true or false  
true  
> █
```

Sample commands has been executed: this is the basic command of the lua.

```
~/code via 🌙 v5.4.4 via 🐙 v3.10.4 took 6s  
> cat temp.lua  
print("Hello, World")  
  
~/code via 🌙 v5.4.4 via 🐙 v3.10.4  
> lua temp.lua  
Hello, World  
  
~/code via 🌙 v5.4.4 via 🐙 v3.10.4  
> █
```

## 4.Comparison of various programming Languages with Lua:

### 4.1:Comparison of various programming languages:

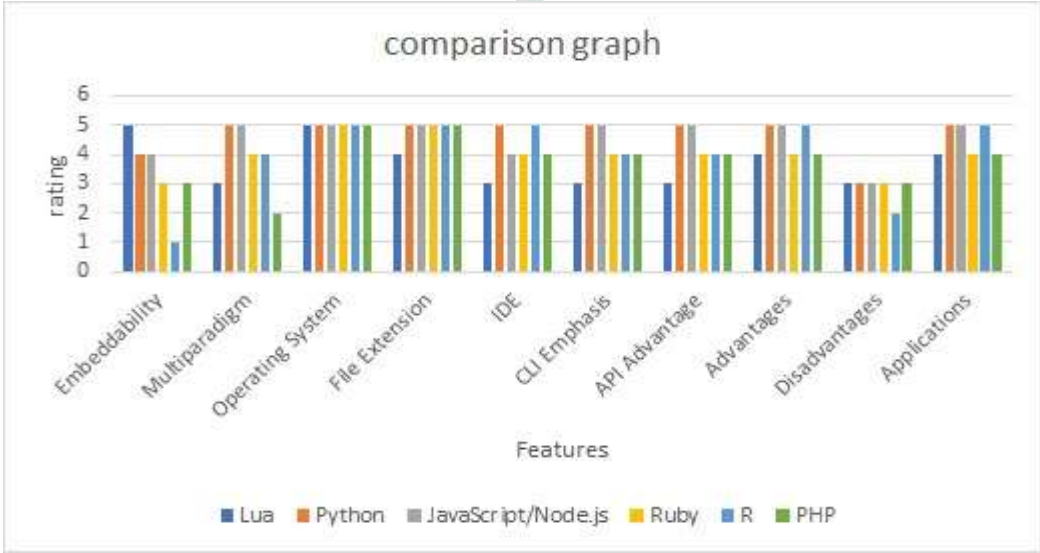
Languages/Parameter	Lua	python	JavaScript/Node.js	Ruby	R	PHP
embed ability	Lua is well-known for its embeddability and is often used as a scripting language in various applications, especially in game development.	Python is embeddable in applications through tools like CPython API.	JavaScript is embedded in web browsers, and Node.js allows it to be used for server-side development.	Ruby can be embedded in Application.	R is typically used as a standalone statistical computing language.	PHP is embedded in HTML and is primarily used for server-side scripting.
Multiparadigm	Lua is primarily a procedural language but supports some features of object-oriented programming.	Python supports object-oriented, imperative, and functional programming paradigms.	JavaScript is a multi-paradigm language with a strong emphasis on event-driven, asynchronous programming.	Ruby is object oriented, reflective and dynamic in nature.	R is primarily used for statistical computing and graphics.	PHP is a server-side scripting language with support for object-oriented programming.
Operating System	Lua is platform-independent.	Python is platform-independent	Platform-independent for browser-based JavaScript; Node.js is cross-platform.	Platform independent.	Platform independent.	Platform independent.
File Extension	Lua scripts typically use the ".lua" file extension.	The file extension are py, .pyc, .pyd, .pyi, .pyo, .pyw, and .pyz.	JavaScript files typically use the ".js" extension.	Ruby scripts typically use the ".rb" file extension.	The file extension is supposed to be .R for R language.	The file extension typically is .php
IDE	Lua doesn't have a standard IDE but can be used with editors like VSCode or dedicated IDEs like ZeroBrane Studio.	Python has a variety of IDEs, including PyCharm, VSCode, and Jupyter Notebooks.	IDEs like VSCode, WebStorm are commonly used.	RubyMine, Atom, VSCode are commonly used.	RStudio is a popular IDE for R.	PhpStorm, VSCode, and others are commonly used.
Operators						
emphasis on CLI/shell.	Lua has a minimalistic command-line interface.	Python has a powerful command-line interface.	Node.js has a powerful command-line interface.	Ruby has a command-line interface.	R has a command-line interface.	PHP has a command-line interface.
API	Lua is often	Extensive	Node.js has a	Ruby is	R is widely	PHP is widely



	used for scripting in game engines like Unity3D.	libraries and frameworks, making it versatile for various applications .	powerful command-line interface.	used with popular web frameworks like Ruby on Rails.	used in data analysis and statistics.	used in web development.
Advantage	Lightweight and easy to embed. Well-suited for scripting in gaming and embedded systems.	Extensive standard library. Versatile and widely used for web development, data science, and more.	Ubiquitous in web development.  Asynchronous programming with Node.js.	Elegant and readable syntax.  Ruby on Rails is a powerful web framework.	Rich statistical libraries.  Strong in data visualization .	Designed for web development.  Large community and extensive documentation.
Disadvantage	Limited standard library compared to some other languages.	Slower execution compared to lower-level languages in certain scenarios.	Single-threaded nature can be a limitation in certain scenarios.	May not be as performant as some other languages.	May not be as versatile for general-purpose programming.	Some aspects of the language may be considered inconsistent.
Application	Game scripting (e.g., in game engines like Unity). Embedded systems.	Web development (Django, Flask). Data science and machine learning (NumPy, TensorFlow ). Automation and scripting.	Web development (frontend and backend with Node.js).  Server-side applications.	Web development (Ruby on Rails).  Scripting and automation.	Data analysis and statistics.	Server-side scripting (web development).  Building dynamic web applications.

Table No.2: Comparison of various programming language with lua.

4.2:Graphical representation of lua and various programming languages



## Graph no. 2 : comparison graph

It give the graphical representation of the various feature of the programming language with the lua. It shows that lua is more efficient in term of embeddability with the others. Lua is totally compatible with the lua operating system and all the languages are compatible with it operating systems that's why all are has the same rating in the above graph. The lua is a terminal based language then it not support the api totally or as compare to python and javascript.

## 5:Results and Discussions

This research is a unique and powerful framework for Linux package management, designed with flexibility and customization in mind.

1. It allows users to implement it in their own projects, add their own features and functionalities, and even change its name.
2. This technique offers the ability for users to add their own databases, allowing for a more personalized and efficient package management experience.
3. It takes the concept of mirrors a step further by allowing users to change the entire database if they need to.
4. 4This is unique advanced scheme using Lua we implemented and successful to use Linux platform,along with options provided.

## 6: Conclusions

The program is more than just a package manager. It's a flexible framework that puts the power Of customization in the hands of the users.

By providing a customizable framework for Linux package management, it aims stem power users and developers a like.

The program can help foster innovation and efficiency in the Linux community by giving users the tool sand freedom to customize their package management experience.

## 7: References

- [1]Package Manager: The Core of a GNU/Linux Distribution
- [2] Arakere, N. K., and Nataraj, C., 1998, "Vibration of High-Speed Spur Gear Webs," ASME Journal of Vibration Acoustics, 120(3), pp.791–800.
- [3] Stewart, R. M., 1977, "Some Useful Data Analysis Techniques for Gearbox Diagnostics, "Proceedings of the Meeting on the Application of Time Series Analysis, Proceeding Paper,
- [4] ISVR, A Comparative Study of various Linux Package-Management Systems, University of Southampton ,Southampton, UK.
- [5] Kong, D. W., 2008, "Research on the Dynamics and Fault Diagnosis of the Large Gear Transmission Systems,"Ph.D., thesis,JiLin University,Changchun,China.
- [6] J. F. Curtis, (Ed.), *Processes and Disorders of Human Comm-unication. IEEE standard*, New York: Harperand Row, 1978.