Linux and HPC Tutorials: A Practical Guide for Beginners

sbilmis

December 27, 2024

Contents

Beginner-Friendly Tutorial: Installing Software Locally on an HPC System

Introduction

In most HPC systems, users do not have **root** privileges. This tutorial focuses on installing software **locally** in your **home** directory **without root access**.

While the examples are based on Rocky Linux, a popular distribution in HPC environments, the steps outlined here will work on most Linux distributions with little or no modification.

It's important to note that commands requiring root privileges, such as `yum install`, `dnf install`, or `sudo apt-get install`, are not available to regular users on HPC systems. When consulting software documentation, focus on sections describing how to install from source or methods for user-level installation, as these do not require administrative access.

Why Root Privileges Are Restricted

- HPC systems are **shared environments with multiple users** running jobs simultaneously.
- Restricting root access ensures **stability, security, and integrity** of the shared system.
- Users are encouraged to install their software locally **to avoid affecting global configurations**.

Installing Software Locally

Step 1: Prepare a Directory for Local Installations

To install software locally, you first need a place to store the files. The '~/.local' directory is commonly used because it follows standard Linux conventions and helps keep your files organized.

The command below creates two directories:

- '~/.local/bin': This is where you will store executable files (programs you can run directly from the terminal).
- '~/.local/src': This is where you will store the source code for the software you are installing.

Here is the command to create these directories:

```
mkdir -p ~/.local/bin ~/.local/src
```

- What this command does:
  - 'mkdir': Stands for **make directory**. It is used to create new directories in your file system.
  - '-p': Ensures that the command creates the entire directory structure, even if some parts (like '~/.local') don't exist yet. If the directories already exist, it won't cause an error.
  - '~/.local/bin' and '~/.local/src': These are the directories being created. The '~' symbol refers to your home directory, so '~/.local' is a hidden folder inside your home directory.
- Viewing Hidden Files The '~/.local' directory is hidden by default because it starts with a '.' (dot). Hidden files and directories do not appear in the output of the regular 'ls' command. To view them, use the '-a' option with 'ls':

  ```
  ls -a
  ```

  This command will list all files and directories, including hidden ones, in the current directory. You can use it to confirm that '~/.local' and its subdirectories have been created successfully.

Step 2: Modify the PATH Variable

The system needs to know where to find the software you install locally. This is achieved by updating the **'PATH'** variable, which tells your shell where to look for executable files.

- What is the PATH Variable?
  - The 'PATH' variable is an environment variable in Linux that contains a list of directories. These directories are searched, in order, whenever you type a command in the terminal.
  - For example, when you type `ls`, the system looks in the directories listed in 'PATH' to find the `ls` program.

  By adding `$HOME/.local/bin` to the 'PATH', you allow the shell to find and execute programs stored in your local `bin` directory.

- What is the $\text{LD}_{\text{LIBRARYPATH}}$ Variable? Shared libraries are files with the '.so' (shared object) extension that contain code and functions used by multiple programs. Instead of including this code directly in each program, shared libraries allow programs to "share" the same code, which saves memory and simplifies updates.

  For example, many programs rely on common libraries like 'libc' (C standard library). Rather than embedding 'libc' in every program, the system provides it as a shared library that all programs can access.

  The `LD_LIBRARY_PATH` variable tells the system where to look for these shared libraries when a program is executed. If the system cannot find the required libraries in the standard locations (like '/usr/lib' or '/lib'), it will search in the directories listed in `LD_LIBRARY_PATH`.

  By adding '~/.local/lib' to `LD_LIBRARY_PATH`, you ensure that any shared libraries installed locally in your home directory can be found and used by your programs.

- Why is This Important for Local Installations? When you install software locally, it may come with its own shared libraries stored in '~/.local/lib'. If these libraries are not in a directory that the system searches by default, the program may fail to run, displaying errors like:

  ```
  error while loading shared libraries: libexample.so: cannot open shared object fil
  ```

Setting `LD_LIBRARY_PATH` to include '~/.local/lib' resolves this issue by explicitly telling the system to search for shared libraries in your local directory.

  - Example Suppose you install a program 'myprogram' that depends on a shared library 'libexample.so' located in '~/.local/lib'. Without setting `LD_LIBRARY_PATH`, running 'myprogram' might result in an error. After adding the following line to your shell configuration file:

    ```
    export LD_LIBRARY_PATH=$HOME/.local/lib:$LD_LIBRARY_PATH
    ```

    And reloading it with:

    ```
    source ~/.bashrc
    ```

    The system will now find libexample.so in ~/.local/lib, allowing myprogram to run successfully.

- Steps to Update the PATH Variable

  1. Open your shell configuration file:
     - **For Bash users:** Open `.bashrc` in a text editor.
     - **For Zsh users:** Open `.zshrc`.
     - **Optional:** Some systems also use `.bash_profile`. If it exists, changes to `.bashrc` might not take effect unless explicitly sourced from `.bash_profile`. To ensure changes apply, you can include the following in your `.bash_profile`:

       ```
       if [ -f ~/.bashrc ]; then
           source ~/.bashrc
       fi
       ```

  2. Add the following lines to the configuration file:

     ```
     export PATH=$HOME/.local/bin:$PATH
     export LD_LIBRARY_PATH=$HOME/.local/lib:$LD_LIBRARY_PATH
     ```

     - What this command does:**
       * `export`: Marks a variable to be available to child processes started from the terminal.

    * `PATH=$HOME/.local/bin:$PATH`: Prepends `.local/bin` to the current `PATH`. This ensures your locally installed executables are prioritized over system-wide ones if they have the same name.
    * `LD_LIBRARY_PATH=$HOME/.local/lib:$LD_LIBRARY_PATH`: Prepends `.local/lib` to the current library search path.

  1. Reload your shell configuration: After saving your changes to `.bashrc` (or `.zshrc`), apply them to the current shell session by running:

     ```
     source ~/.bashrc
     ```

  – What this command does:**
    * The `source` command reads and executes the contents of the file (`.bashrc` in this case) in the current shell session. This ensures the changes take effect without needing to log out and back in.

- Verifying the Changes You can check if the `PATH` and `LD_LIBRARY_PATH` variables were updated successfully by running:

  ```
  echo $PATH
  echo $LD_LIBRARY_PATH
  ```

  These commands will display the values of the respective environment variables. Look for `$HOME/.local/bin` in `PATH` and `$HOME/.local/lib` in `LD_LIBRARY_PATH`.

- Why Use ~/.bashrc Instead of ~/.bash$_{\text{profile}}$?

  – `.bashrc` is executed for non-login interactive shells, which is what you typically use when opening a terminal.

  – `.bash_profile` is executed for login shells. On many systems, `.bash_profile` sources `.bashrc`, ensuring the settings apply in all cases.

  – If you find your changes in `.bashrc` aren't taking effect, check if your `.bash_profile` sources it as shown earlier.

  With these steps completed, your system is now configured to recognize and execute programs installed in your local directory.

Step 3: Understand the Basics of Building Software from Source

When installing software from source, the process typically follows three key steps: **'configure'**, **'make'**, and **'make install'**. These steps are designed to prepare, compile, and install the software on your system.

- Why Build Software from Source?
  - **Flexibility:** You can customize the installation to suit your specific needs. For example:
    - Imagine you are installing a program that supports optional features like GUI tools or database support. When building from source, you can enable or disable these features using options in the 'configure' step. For instance:

      ```
      ./configure --prefix=$HOME/.local --enable-gui --disable-database
      ```

      This command installs the program with GUI support but without database functionality, tailored to your requirements.
    - Such customizations are often not possible with pre-built binaries, which are typically compiled with default settings.
  - **Portability:** Source code can be compiled and adapted to work on different systems, as long as the necessary tools (like compilers) and dependencies are present. Unlike pre-built binaries, which are often specific to a particular operating system or hardware architecture, source code is more flexible. For example:
    - A binary compiled for **Ubuntu** may not work on **CentOS** due to differences in system libraries or architecture.
    - However, source code can be compiled on both systems to produce binaries that work natively.
  - **Control:** You can install software in a **specific directory**, such as '~/.local', without requiring root privileges.
- The Three Steps in Detail:
  1. **'configure': Preparing the Build**
     - The 'configure' script sets up the software for compilation. It checks your system to ensure all required dependencies and libraries are available.
     - It also allows you to customize the build by using options like '–prefix' to specify the installation directory.

- Example:

```
./configure --prefix=$HOME/.local
```

  - **What happens during ‘configure‘:**
  - The script checks for compilers (e.g., ‘gcc‘), libraries, and other tools required to build the software.
  - It generates configuration files and Makefiles that tell the ‘make‘ command how to build the software.
- **Common issues during ‘configure‘:**
  - Missing dependencies: The script may fail if required libraries are not installed. Use tools like **‘module load‘** (on HPC systems) or install missing dependencies locally. We will have an example for this below.

It is important to read the output of the code for missing dependencies and errors.

1. **‘make‘: Compiling the Source Code**
   - The ‘make‘ command uses the Makefile generated by ‘configure‘ to compile the source code into binary executables.
   - Example:

```
make
```

   - **What happens during ‘make‘:**
     - The source code files (usually written in C, C++, etc.) are translated into machine code by a compiler.
     - This step may **take time depending on the size of the software** and your system's performance.
   - **Common issues during ‘make‘:**
     - Compiler errors: These can happen if the source code is not compatible with your system or if dependencies are missing.
2. **‘make install‘: Installing the Compiled Software**
   - The ‘make install‘ command copies the compiled binaries, libraries, and other files to the specified installation directory (e.g., ‘~/.local/bin‘).
   - Example:

```
make install
```

   - **What happens during ‘make install‘:**

    - Binaries are copied to '~/.local/bin' (or the directory specified with '–prefix' during 'configure').
    - Shared libraries and other resources are placed in the appropriate locations, such as '~/.local/lib'.

- Summary of the Process:

    1. **'configure':** Prepares the source code for your system and customizes the installation.
    2. **'make':** Builds the software by compiling the source code.
    3. **'make install':** Installs the built software in the target directory.

- Additional Notes:

    - **Customizing the Installation:**
        - Use **'./configure –help'** to see all available options. For example, you can disable optional features or specify non-default library paths.
    - **Verifying the Installation:**
        - After running **'make install'**, check that the program is accessible by running:

          ```
          $HOME/.local/bin/<program-name> --version
          ```
    - **Cleaning Up:**
        - To remove compiled files and start fresh, use:

          ```
          make clean
          ```

    By understanding these steps, you'll be able to install a wide variety of software from source, even on systems where you don't have root access.

Example: Installing Software

Installing 'inxi'

1. **Download the Script**

   ```
   cd ~/.local/src
   wget -O inxi https://github.com/smxi/inxi/archive/master.zip
   ```

2. **Extract and Install**

```
unzip inxi-master.zip
cd inxi-master
cp inxi $HOME/.local/bin/
chmod +x $HOME/.local/bin/inxi
```

3. **Verify Installation**

```
inxi --version
```

Installing ‘ncdu‘ (with ‘configure‘, ‘make‘, and ‘make install‘)

1. **Download the Source**

```
wget https://dev.yorhel.nl/download/ncdu-1.21.tar.gz
tar -xvzf ncdu-1.21.tar.gz
cd ncdu-1.21
```

2. **Build and Install**

```
./configure --prefix=$HOME/.local
make
make install
```

3. **Verify Installation**

```
ncdu --version
```

Example with HPC Modules: Installing Software Dependencies

Many HPC systems provide software modules that allow loading dependencies without installation:

1. List available modules:

```
module avail
```

2. Load a required module:

```
module load gcc
```

3. Build your software using the loaded module.

Useful Tools: ‘tldr‘ and ‘cheat‘

- **‘tldr‘:** Provides simplified, community-driven examples for commands. Install it locally:

```
pip install --user tldr
tldr tar
```

- **‘cheat‘:** Allows creating and viewing custom cheat sheets.

```
pip install --user cheat
cheat tar
```

Removing Locally Installed Applications

To remove software:

1. For binaries:

```
rm $HOME/.local/bin/<binary>
```

2. For source-built software:

```
make uninstall
```

Troubleshooting Tips

- **Command not found:** Verify that ‘~/.local/bin‘ is in your ‘PATH‘.
- **Missing dependencies:** Use modules or install dependencies locally.
- **Permission errors:** Ensure your home directory permissions are correct and use ‘chmod +x‘ for executable files.

Conclusion

Installing software locally empowers you to work efficiently in an HPC environment without requiring root privileges. With practice, building software from source and managing your tools will become second nature.

Beginner-Friendly Tutorial: Installing Software Locally on an HPC System

Introduction

High-Performance Computing (HPC) systems are powerful tools that enable users to perform complex computations efficiently. However, these systems are shared resources, and to maintain their stability and security, users typically do not have root privileges. This means that installing software system-wide using package managers like ‘yum‘ or ‘dnf‘ is not possible for regular users. Instead, users are encouraged to install software locally within their home directories.

This tutorial will guide you through the process of installing software locally from source code on an HPC system running **Rocky Linux**. We will use examples such as **inxi** (a binary package) and **ncdu** (which requires configuration and compilation). Additionally, we will explore using modules in an HPC environment, alternatives to traditional man pages, and methods to remove locally installed applications.

Why Root Privileges Are Restricted

- **Shared Resources:** HPC systems are utilized by multiple users simultaneously. Allowing root access could lead to conflicts and resource mismanagement.
- **System Integrity:** Limiting root access prevents accidental or malicious changes that could compromise the system's stability and security.
- **Encouraging Best Practices:** Users are motivated to install software locally, fostering a better understanding of software management without affecting the global environment.

Installing Software Locally

To install software without root privileges, follow these general steps:

1. **Prepare a Directory for Local Installations** It is recommended to use ‘~/.local‘ for local installations. This directory adheres to the [XDG Base Directory Specification](`https://specifications.freedesktop.org/basedir-spec/basedir-spec-latest.html`), ensuring compatibility with many tools.

“‘bash mkdir -p ~/.local/bin ~/.local/src “‘

- ‘~/.local/bin‘: Directory where executable binaries will be stored.
- ‘~/.local/src‘: Directory for storing source code of applications.

2. **Modify the PATH and $\text{LD}_{\text{LIBRARYPATH}}$ Variables** To ensure the system recognizes the locally installed software, update your environment variables by adding the local ‘bin‘ and ‘lib‘ directories to ‘PATH‘ and ‘$\text{LD}_{\text{LIBRARYPATH}}$‘, respectively.

   Add the following lines to your shell configuration file (e.g., ‘~/.bashrc‘ or ‘~/.zshrc‘):

   “‘bash export PATH=$HOME/.local/bin:$PATH export $\text{LD}_{\text{LIBRARYPATH}}$=$HOME/.local/lib:$LI
   “‘

   - **PATH:** Specifies directories where executable programs are located. Adding ‘~/.local/bin‘ allows the shell to locate your locally installed executables.
   - **$\text{LD}_{\text{LIBRARYPATH}}$:** Specifies directories where shared libraries are searched for first. Adding ‘~/.local/lib‘ ensures that any locally installed libraries are found by applications.

   Reload Your Shell Configuration:

   “‘bash source ~/.bashrc “‘

   This command applies the changes without requiring you to restart the terminal.

3. **Understanding Common Installation Commands** When installing software from source, you’ll frequently encounter the following commands:

   - **‘configure‘:** A script that prepares the build system for your specific environment. It checks for necessary dependencies and configures makefiles accordingly.
   - **‘make‘:** Compiles the source code into executable binaries based on the configurations set by ‘configure‘.
   - **‘make install‘:** Installs the compiled binaries and other necessary files into the specified directories (e.g., ‘~/.local‘).

Example 1: Installing Inxi (Binary Package)

What is Inxi?

Inxi is a powerful command-line system information tool that provides detailed information about your system's hardware and software configurations.

Installation Steps

1. **Download the Inxi Script** Navigate to the source directory and download the inxi script.

   "'bash cd ~/.local/src wget -O inxi `https://github.com/smxi/inxi/raw/master/inxi` "'

   - **Explanation:** 'wget' downloads the inxi script directly from its GitHub repository and saves it as 'inxi' in the '~/.local/src' directory.

2. **Install the Inxi Script** Copy the script to the local 'bin' directory and make it executable.

   "'bash cp inxi ~/.local/bin/ chmod +x ~/.local/bin/inxi "'

   - **'cp inxi ~/.local/bin/':** Copies the inxi script to '~/.local/bin' so it can be executed from anywhere.
   - **'chmod +x ~/.local/bin/inxi':** Grants execute permissions to the script.

3. **Verify the Installation**

   "'bash inxi –version "'

   - **Explanation:** This command checks if inxi is accessible and displays its version, confirming a successful installation.

Example 2: Installing ncdu (Source Compilation)

What is ncdu?

'ncdu' (NCurses Disk Usage) is a disk usage analyzer with a text-based interface, ideal for identifying large files or directories on an HPC system.

Installation Steps

1. **Download the Source Code**

   “‘bash cd ~/.local/src wget `https://dev.yorhel.nl/download/ncdu-1.21.tar.gz` “‘

   - **Explanation:** Downloads the ‘ncdu‘ source tarball to ‘~/.local/src‘.

2. **Extract the Archive**

   “‘bash tar -xzvf ncdu-1.21.tar.gz cd ncdu-1.21 “‘

   - **‘tar -xzvf‘:** Extracts the compressed tarball.
   - **‘cd ncdu-1.21‘:** Navigates into the extracted source directory.

3. **Configure the Build Environment**

   “‘bash ./configure –prefix=$HOME/.local “‘

   - **Explanation:** Prepares the build system to install ‘ncdu‘ into the ‘~/.local‘ directory. The ‘–prefix‘ flag specifies the installation directory.

4. **Compile the Source Code**

   “‘bash make “‘

   - **Explanation:** Compiles the source code into executable binaries based on the configurations set by the ‘configure‘ script.

5. **Install the Compiled Software**

   “‘bash make install “‘

   - **Explanation:** Installs the compiled binaries and necessary files into the specified directories within ‘~/.local‘.

6. **Verify the Installation**

   “‘bash ncdu –version “‘

   - **Explanation:** Checks if ‘ncdu‘ is accessible and displays its version, confirming a successful installation.

Example 3: Installing mc (Midnight Commander)

What is Midnight Commander (mc)?

Midnight Commander is a text-based file manager that provides a user-friendly interface for navigating and managing files directly from the command line. It includes features like directory browsing, file manipulation, and file viewing.

Installation Steps

1. **Download the Source Code**

   “‘bash cd ~/.local/src wget `http://ftp.midnight-commander.org/mc-4.8.27.tar.xz` “‘

   - **Note:** Replace ‘4.8.27‘ with the latest version number if necessary.

2. **Extract the Archive**

   “‘bash tar -xJvf mc-4.8.27.tar.xz cd mc-4.8.27 “‘

   - **‘tar -xJvf‘:** Extracts the ‘.tar.xz‘ compressed tarball.

3. **Configure the Build Environment**

   “‘bash ./configure –prefix=$HOME/.local “‘

   - **Explanation:** Sets up the build system to install ‘mc‘ into ‘~/.local‘.

4. **Compile the Source Code**

   “‘bash make “‘

5. **Install the Compiled Software**

   “‘bash make install “‘

6. **Verify the Installation**

   “‘bash mc –version “‘

   - **Explanation:** Checks if ‘mc‘ is accessible and displays its version, confirming a successful installation.

Using Modules in an HPC Environment

Many HPC systems use environment modules to manage software and their dependencies. Modules allow users to dynamically modify their shell environment to access different software versions without interfering with each other.

Example: Loading a Module and Installing Software

1. **List Available Modules**

   “‘bash module avail “‘

   - **Explanation:** Displays all available modules that can be loaded.

2. **Load a Specific Module**

   “‘bash module load gcc/9.3.0 “‘

   - **Explanation:** Loads GCC version 9.3.0, which may be required for compiling certain software.

3. **Verify Loaded Modules**

   “‘bash module list “‘

   - **Explanation:** Lists all currently loaded modules in your environment.

4. **Install Software Using Loaded Modules**

   Follow the standard installation steps (download, extract, configure, make, make install) while the necessary modules are loaded to ensure all dependencies are met.

Understanding TLDR and Cheat Commands

For users who find traditional man pages (‘man‘) overwhelming or verbose, ‘tldr‘ and ‘cheat‘ provide simplified and practical command examples.

1. **tldr**

   - **What is tldr?** A collection of simplified and community-driven man pages.

- **Installation:**
  “‘bash
  pip install tldr “‘
- **Usage Example:**
  “‘bash tldr tar “‘
  - **Explanation:** Displays concise usage information and common examples for the ‘tar‘ command.

2. **cheat**

   - **What is cheat?** A tool that allows users to create and view interactive cheat sheets for various commands.
   - **Installation:**
     “‘bash
     pip install cheat “‘
   - **Usage Example:**
     “‘bash cheat tar “‘
     - **Explanation:** Shows practical examples and usage scenarios for the ‘tar‘ command.

Unzipping and Untarring Software Archives

When installing software from compressed archives, understanding how to extract them is crucial.

1. **Unzip (‘.zip‘ files)**

   “‘bash unzip filename.zip -d $\text{destination}_{\text{directory}}$ “‘

   - **‘-d $\text{destination}_{\text{directory}}$‘:** Specifies the directory to extract files into.

2. **Untar (‘.tar.gz‘ or ‘.tar.xz‘ files)**

   - **For ‘.tar.gz‘ files:**
     “‘bash tar -xzvf filename.tar.gz -C $\text{destination}_{\text{directory}}$ “‘
   - **For ‘.tar.xz‘ files:**
     “‘bash tar -xJvf filename.tar.xz -C $\text{destination}_{\text{directory}}$ “‘

- **Explanation:**
  - ‘-x‘: Extract files.
  - ‘-z‘: Filter the archive through gzip.
  - ‘-J‘: Filter the archive through xz.
  - ‘-v‘: Verbosely list files processed.
  - ‘-f‘: Use archive file.
  - ‘-C destination$_{\text{directory}}$‘: Change to directory before performing any operations.

Removing Locally Installed Applications

If you need to remove a locally installed application, follow these steps:

1. **Delete Executable Binaries**

   “‘bash rm ~/.local/bin/application$_{\text{name}}$ “‘

   - **Explanation:** Removes the executable from the ‘bin‘ directory.

2. **Delete Library Files (if any)**

   “‘bash rm -r ~/.local/lib/application$_{\text{name}}$ “‘

   - **Explanation:** Removes any associated libraries.

3. **Delete Source Code (optional)**

   “‘bash rm -r ~/.local/src/application$_{\text{sourcedirectory}}$ “‘

   - **Explanation:** Removes the source code directory if you no longer need it.

4. **Update Environment Variables (if necessary)**

   If you added specific environment variables for the application, remove or comment them out from your shell configuration file (‘~/.bashrc‘ or ‘~/.zshrc‘), then reload the configuration:

   “‘bash source ~/.bashrc “‘

Troubleshooting Tips

- **Problem:** ‘command not found‘ after installing the software.
  - **Solution:** Ensure the ‘PATH‘ variable includes ‘~/.local/bin‘ by running ‘echo $PATH‘. If not, revisit the **Modify the PATH and LD$_{\text{LIBRARYPATH}}$ Variables** section.
- **Problem:** Missing dependencies during installation.
  - **Solution:** Use modules provided by the HPC center (e.g., ‘module load gcc‘) or install dependencies locally following the same installation procedures.
- **Problem:** Permissions errors while copying or executing files.
  - **Solution:** Ensure you have write permissions in your home directory. Use ‘chmod +x‘ to make scripts executable if necessary.
- **Problem:** Compilation errors during ‘make‘.
  - **Solution:** Verify that all required development libraries and tools are installed. Load necessary modules using ‘module load‘.
- **Problem:** Issues with library paths or missing shared libraries.
  - **Solution:** Ensure that ‘LD$_{\text{LIBRARYPATH}}$‘ includes directories where required libraries are installed. You may need to add ‘~/.local/lib‘ if not already present.

Conclusion

Installing software locally on an HPC system empowers you to customize your computing environment without requiring administrative privileges. By using your home directory and properly configuring environment variables, you can access a wide range of tools tailored to your needs. Understanding the installation process—from downloading and extracting source code to compiling and installing—ensures you can manage software effectively. Additionally, leveraging tools like ‘tldr‘ and ‘cheat‘ can simplify command usage, enhancing your productivity. Remember to maintain your local installations and remove any unnecessary applications to keep your environment clean and efficient.

Happy Computing!

Beginner-Friendly Tutorial: Installing Software Locally on an HPC System

Introduction

In most HPC systems, users do not have root privileges, and in this case, the system is running **Rocky Linux**. Root privileges, such as running 'yum install' or 'dnf install' commands, are not possible for regular users. This ensures the stability and security of the shared computing environment.

When searching for documentation on how to install software, avoid instructions that require root access (e.g., using package managers like 'yum' or 'dnf'). Instead, focus on guides for installing software from source or using user-level installation methods.

This tutorial will guide you through the process of installing software locally, using the example of installing **inxi package** from source.

Why Root Privileges Are Restricted

- HPC systems are shared resources where multiple users run jobs simultaneously.
- Limiting root access ensures system integrity and prevents accidental or malicious changes.
- Users are encouraged to install software locally without impacting the global environment.

Installing Software Locally

1. **Prepare a Directory for Local Installations** As a general practice, it is recommended to use '~/.local' for local installations. This directory follows standard conventions and is automatically recognized by many tools. However, note that '~/.local' is a hidden directory. To view it, you need to use the '~ls -a~' command, as it will not appear in a normal 'ls' listing.

   Create a directory in your home folder to store installed software:

   ```
   mkdir -p ~/.local/bin ~/.local/src
   ```

2. **Modify the PATH Variable** Update your environment variables to include the local installation directory in your PATH. Add the following lines to your shell configuration file (e.g., '~/.bashrc' or '~/.zshrc'):

```
export PATH=~/.local/bin:$PATH
export LD_LIBRARY_PATH=~/.local/lib:$LD_LIBRARY_PATH
```

Reload your shell configuration:

```
source ~/.bashrc
```

test

Example: Installing Inxi from Source

1. **Download the Source Code** Navigate to the source directory and download the inxi source code:

   ```
   cd ~/.local/src
   wget -O inxi https://github.com/smxi/inxi/archive/master.zip
   ```

2. **Extract the Archive** Extract the downloaded file:

   ```
   unzip inxi
   cd inxi-master
   ```

3. **Install the Script** Copy the inxi script to the local ‘bin‘ directory:

   ```
   cp inxi ~/.local/bin/
   chmod +x ~/.local/bin/inxi
   ```

4. **Verify the Installation** Confirm that inxi is installed and accessible:

   ```
   inxi --version
   ```

Additional Examples: mc and ncdu

Midnight Commander (mc)

- **What is it?** Midnight Commander is a text-based file manager that provides a user-friendly interface for navigating and managing files directly from the command line. It includes features like directory browsing, file manipulation, and file viewing.
- **Source Code:**

  - Official Website: https://midnight-commander.org/
  - Git Repository: https://github.com/MidnightCommander/mc
  - Release Tarballs: https://ftp.midnight-commander.org/

wget http://ftp.midnight-commander.org/mc-4.6.1.tar.gz

- **Installation Steps:** After downloading the source code from the above links, follow the standard build process:

```
./configure --prefix=$HOME/.local
make
make install
```

ncdu

- **What is it?** ncdu (NCurses Disk Usage) is a disk usage analyzer with a text-based interface, ideal for identifying large files or directories on an HPC system.
- **Source Code:**
  - Official Website: https://dev.yorhel.nl/ncdu
  - Git Repository: https://g.blicky.net/ncdu.git

wget https://dev.yorhel.nl/download/ncdu-1.21.tar.gz

- **Installation Steps:** Clone the repository or download the source tarball, then build and install:

```
./configure --prefix=$HOME/.local
make
make install
```

Troubleshooting Tips

- **Problem:** ‘command not found‘ after installing the software.
  - **Solution:** Ensure the PATH variable includes ‘~/.local/bin‘ by running ‘echo $PATH‘.
- **Problem:** Missing dependencies during installation.
  - **Solution:** Use modules provided by the HPC center or install dependencies locally.
- **Problem:** Permissions errors while copying or executing files.
  - **Solution:** Ensure you have write permissions in your home directory and use ‘chmod +x‘ to make scripts executable.

Conclusion

Installing software locally on an HPC system is straightforward when you understand the process. By using your home directory and updating your PATH, you can have the flexibility to use the tools you need without requiring administrative access. For consistency and compatibility, it is recommended to use ‘~/.local‘ for local installations. Remember that ‘~/.local‘ is hidden, and you can view it with the ‘ls -a‘ command.

Beginner-Friendly Tutorial: Installing Software Locally on an HPC System

Introduction

In most HPC systems, users do not have root privileges, and in this case, the system is running **Rocky Linux**. Root privileges, such as running ‘yum install‘ or ‘dnf install‘ commands, are not possible for regular users. This ensures the stability and security of the shared computing environment.

When searching for documentation on how to install software, avoid instructions that require root access (e.g., using package managers like ‘*yum*‘, ‘*dnf*‘ or **‘apt-get‘**). Instead, focus on guides for installing software from source or using user-level installation methods.

This tutorial will guide you through the process of installing software locally, using the example of installing **inxi** from source.

Why Root Privileges Are Restricted

- HPC systems are shared resources where multiple users run jobs simultaneously.
- Limiting root access ensures system integrity and prevents accidental or malicious changes.
- Users are encouraged to install software locally without impacting the global environment.

Installing Software Locally

1. **Prepare a Directory for Local Installations** As a general practice, it is recommended to use '~/.local' for local installations. This directory follows standard conventions and is automatically recognized by many tools. However, note that '~/.local' is a hidden directory. To view it, you need to use the 'ls -a' command, as it will not appear in a normal 'ls' listing.

   Create a directory in your home folder to store installed software:

   ```
   mkdir -p ~/.local/bin ~/.local/src
   ```

2. **Modify the PATH Variable** Update your environment variables to include the local installation directory in your PATH. Add the following lines to your shell configuration file (e.g., '~/.bashrc' or '~/.zshrc'):

   ```
   export PATH=~/.local/bin:$PATH
   export LD_LIBRARY_PATH=~/.local/lib:$LD_LIBRARY_PATH
   ```

   Reload your shell configuration:

   ```
   source ~/.bashrc
   ```

Example: Installing Inxi from Source

1. **Download the Source Code** Navigate to the source directory and download the inxi source code:

   ```
   cd ~/.local/src
   wget -O inxi https://github.com/smxi/inxi/archive/master.zip
   ```

2. **Extract the Archive** Extract the downloaded file:

```
unzip master.zip
cd inxi-master
```

3. **Install the Script** Copy the inxi script to the local ‘bin‘ directory:

```
cp inxi ~/.local/bin/
chmod +x ~/.local/bin/inxi
```

4. **Verify the Installation** Confirm that inxi is installed and accessible:

```
inxi --version
```

Troubleshooting Tips

- **Problem:** ‘command not found‘ after installing the software.
  - **Solution:** Ensure the PATH variable includes ‘~/.local/bin‘ by running ‘echo $PATH‘.
- **Problem:** Missing dependencies during installation.
  - **Solution:** Use modules provided by the HPC center or install dependencies locally.
- **Problem:** Permissions errors while copying or executing files.
  - **Solution:** Ensure you have write permissions in your home directory and use ‘chmod +x‘ to make scripts executable.

Conclusion

Installing software locally on an HPC system is straightforward when you understand the process. By using your home directory and updating your PATH, you can have the flexibility to use the tools you need without requiring administrative access. For consistency and compatibility, it is recommended to use ‘~/.local‘ for local installations. Remember that ‘~/.local‘ is hidden, and you can view it with the ‘ls -a‘ command.