Lab 2

Application Management

2.1 Objectives

- Install software ready to use in a specific operating system (binaries)
- Install software starting from source code

2.2 Before you start

It would be good if you can think about these questions, and answer them.

❓ *Which command can you use to download files from a `web` server?*

❓ *How can we list the contents of a tar file?*

❓ *What should we do if the file is gzipped?*

❓ *How can we extract the contents of a tar file?*

❓ *How about a `tar.gz` file? And a `tar.bz2`?*

❓ *How can we create a hardlink to a file?*

How can we create a softlink to a file?

What is the use of the `PATH` environment variable?

2.3 Introduction

The process of installing software on an operating system is essentially the copy of files that make up the application in the appropriate directories and, optionally, the modification of some parameter settings of the application.

2.3.1 Software management systems

Since the introduction of dynamic libraries, before installing a particular software, we have to take care of installing its dependencies. Dependencies consist of other software that has to be installed in advance, because it is used by our new application. To help administration of dependencies, several software management systems have appeared. These systems keep track of the libraries and applications installed, so that they allow easy install, update, and uninstall software.

Specifically, our ASO Linux is based on Debian. Debian organizes its software in a series of compressed files forming a package (packages are similar to `tar.gz` files, actually they are `ar` files). Each package can include binaries, libraries, configuration files, manual pages, other documentation, and moreover, contain information on dependencies with other packages. Debian packages have a `.deb` extension.

2.3.2 The UNIX graphical environment: X-Window

The X-window system (or X11, or X) is a protocol to display graphics, providing a standard toolkit for building graphical user interfaces (GUI). X provides the basic frame of reference, but does not define the user interface. This is left to client programs. Further, X makes use of a client/server model that communicates the X server, locally or networked, with client programs. The server accepts requests for graphical output (windows) and sends back to user the input received from the peripherals (keyboard, mouse, or other).

The X system has no specification on the specifics of the user interface such as: buttons, menus, etc. Instead, the software applications are responsible for the appearance of their windows. To allow that several diverse applications have a similar looking, they usually rely on the services of window managers, and desktop environments.

There are different implementations of the X-window system for Linux (and other UNIX systems). The most common and the one that we use is called X.org. In addition to the X server, window managers and desktop environments also require the installation of their own packages.

Window manager : is responsible for controlling location and appearance of the windows of graphical applications. There are many window managers with different functionalities. Some of them are: `kwin`, `gnome-shell`.

Display Manager : allows to start a new session in the machine. The display manager screen presents the user with a login and password validation. Therefore it performs functions like the `getty` and `login` programs do on character mode terminals. Some common managers are: `XDM` (the X Window Display Manager), `GDM` (Gnome Display Manager) and `SDDM` (KDE Display Manager). The Display Manager is a service that can be started and stopped as the rest of system services using startup scripts, using the `systemctl` command.

Desktop environment : provides a unified interface to the user for applications with graphical icons, tool bars, backgrounds, etc. The desktop environment typically consists of a window manager, a screen manager and its own a set of applications and libraries. The most common desktop environments are KDE and GNOME, but there are many more.

Table 2.1 shows a list of several desktop environments with its corresponding window and display manager

Desktop Environment	Window Manager	Display Manager	Graphical library
GNOME	gnome-shell	GDM	GTK+
KDE	Kwin	SDDM	QT
XfCE	Xfwm4	LightDM	GTK+
LXDE	Openbox	LXDM	QT

Table 2.1: Desktop Environments and its depending window and display managers

2.4 Installation of binary packages

Before getting to the installation of the window manager, let's first understand how package installation works.

2.4.1 Manual installation

We want to install the `make` application into our system. First, we need to get the software to install. The packages you will need are in the ASO Web server.

To access the package you can use the following command:

```
$ wget https://asoserver.pc.ac.upc.edu/aso_public/make_4.4.1-2_amd64.deb
```

Access to the server, and go into the packages directory. Download the appropriate package containing the `make` command.

To install a package `.deb`, use the `dpkg` command (Debian PacKaGe). The following command should work:

```
$ dpkg --install <file.deb>
```

Read the messages that `dpkg` prints during the process and ensure that no problems are reported. You should get used to such kinds of messages.

The `dpkg` command also allows to obtain information about installed packages, and files, and to uninstall packages. Please see the help provided by the `dpkg` command itself and/or its man page, and complete the following table:

Action	**Options**	**Arguments**
Install a package	-i or –install	
Uninstall a package		
Purge a package		
List installed packages		
List files in a package		

Table 2.2: package management with `dpkg`

What is the difference between uninstall and purge a package?

Now, to make things a little bit more interesting, we want to install the `lynx` program (a text-based web browser), you'll see that the application actually has two packages. Download the packages from the ASO web server, and install them with the `dpkg` command. Just for the record, try to install first the `lynx` package (without the `-common`).

Which command did you use?

As you can see, the installation failed, this is because lynx has a *dependency*, which actually is the lynx-common package. So, let's install it with `dpkg`. Now all should be fine and both packages should be properly configured.

Execute the `lynx` command to ensure that it works correctly.

2.4.2 Installation with a package manager

Package managers are useful to facilitate the installation of large applications (which usually have many dependencies) and also make it easier to keep systems up to date.

Debian has a toolset, called APT (Advanced Packaging Tool), an advanced front-end for `dpkg` that you can use to search, download and install software and all its dependencies, and keep the system updated in an easy and convenient way. There are also several graphical front-ends (`Synaptic`, `Adept`, ...) that we will not use in this course.

Configuring the software repositories

First we need to correctly configure the APT repositories from where the manager can get the `.deb` files to install in the system. These repositories can be on remote servers or even other local directories and we can have configured as many of them as we want.

APT configuration files are in */etc/apt*. Inside this directory we will see a file named *sources.list*, change it to match the following contents:

```
deb http://ftp.es.debian.org/debian/ stable main non-free contrib
```

Now, we have to force our system to get the updated list of packages available in the newly configured repositories, and all the information related, i.e., dependencies. This is the appropriate command:

```
# apt update
```

The `apt` tool is also useful, among others, to install, update, and uninstall packages.

Which command (and parameters) will we use to update all our installed packages to the more recent version available?

If you need more information on a specific package (its description, its dependencies, ...) you can use:

```
$ apt info package-name
```

Installation of the X-window system

To handle packages, Debian uses several tools, we've already seen the lowest level application for the task, `dpkg`, however, when applications have dependencies, going package by package and downloading them may be a hassle, so, it is way better to leave the packet manager to perform the task. In Debian we have several, but the most often used are `apt`.

Now, use the `apt` command to install an X server. The package you must install is `x-window-system`. Note also that the `apt` tool installs all the dependencies needed and asks you the questions necessary to configure the X server.

Which command have you used?

In addition to an X server, we need a window manager and a desktop environment. Nowadays this comes as a whole package, and Debian provides a convenient way of installing the whole window system. This is achieved by searching for the right window manager. If we do not know of any, we can search the database package. There are several ways to do that. We will use `apt` command for that. We will first try to get the list of all the packages available in the database.

Which command have you used?

Now, consider that Debian has a group of metapackages [1]. In the case of complete desktops these packages are named `tasks`. Now if we search for all the list of packages starting by *task*.

Which command did you use?

Probably you can use `grep` to filter out the command output. This will help in a lot of commands, since you can just get the output you really need.
As a bonus hint, start of line in `grep` is done using the character `^`.

Now, you can further filter out the output knowing that the desktop package names have the form: `task-[flavor]-desktop`.

Which command did you use?

Another interesting tool of APT is `apt search`. `apt search` searches among the information that the system got from the repositories after being updated. Using `apt search` you can find the desktop environments we have available to be installed in the system.

What command have you used?

Now, select a Desktop Environment to install, and get them for your system using `apt`. Some examples of Desktop Environment to narrow the search:

- KDE → `task-kde-desktop`
- Gnome → `task-gnome-desktop`
- XfCE4 → `task-xfce-desktop`

Which command have you used?

Sometimes the default package configuration is not doing well, or the configuration files of a given package can get damaged by an error. In these cases you will need to reconfigure the package and generate the configuration files again. The APT system uses the `dpkg` command to do this:

```
# dpkg-reconfigure package-name
```

If you have problems with the X system configuration, you can use this command to reconfigure the X server.

2.4.3 Preparing the system to compile software

Now install the following packages: `gcc` (compiler), `libc6-dev` (development libraries) and `firefox`[1].

List here the commands used for the installation:

When you finish, please run the following command:

```
# apt clean
```

What do you think is this command doing?

What is the difference with the command `apt` autoclean?

[1] It could happen that, depending on the Descktop Environment you selected, `firefox` is already installed

2.4.4 Installation of pre-compiled binaries

Sometimes we want to install software that is not (for whatever reason) into a package on our repositories. In this exercise, we are going to install two versions of the Java SDK (JDK).

Download the file corresponding to the `java` installation from our ASO web server. They are in the `aso_public/java` directory. To uncompress the files run tar.

Initially we want to install version 1.25 (`openjdk-25_linux-x64_bin.tar.gz`) into */opt/java1.25*.

? *What commands do you use to uncompress the file?*

? *In which directory did you get the uncompressed files?*

? *Now move the base directory to its final location (prefix): /opt/java1.25*

There you can observe that the installation created a series of directories. Look at the contents and locate where the executable `java` (the one executing the java virtual machine) is. Verify that it is properly installed:

```
$ <BINARY LOCATION> -version
```

Now repeat this step for JREs 1.21, but now let's install it using the package in the Debian repository:

```
$ sudo apt install openjdk-21-jre
```

Now, if we try to find out which java version is the default:

```
$ java -version
```

Now, the easiest way to select which version of java you want to run is to update the `PATH` variable.

Another option is to use `update-alternatives` application present on all Debian systems, but this is out of the scope of this lab, feel free to check it if you want though.

Now, in addition, we want that each version could be directly accessible with the commands java`Version` (e.g., java1.25, java1.21)

? *Which commands do you use to achieve this?*

2.5 Installation from source code

Sometimes we have to install an application directly from the source code, either because the package does not exist in the repositories, or because we want the installation of the application to suit somehow to our system.

As an example, we are going to install a small restricted shell that will be used in other lab sessions. Download the *asosh-0.1.tar.gz* from the sources directory in the ASO FTP server.

A usual place to put the source code is in */usr/src*. Uncompress the source code with the `tar` command in that directory.

Which command did you use?

Look at the contents of the directory with the source code. Usually you will find a script named "configure", that will let you configure parts of the compilation and installation processes (enable / disable parts of the code, choose the installation directory, ...). The specific information about this script can usually be found in the INSTALL and README files.

By default, asosh will be installed in */usr/local*. Run the configure script properly to install it in */usr/local/asosh*.

Which parameters did you use?

Note that the test for required libraries gives an error because they are not installed.

Which is the error reported?

What is the reason for this error?

How did you solve the problem?

Hint: Headers are usually in a separate package. Normally development packages come from libraries, i.e., they start with the word `lib`, and end with the suffix `-dev`.
For example, the development package of libc, as we saw earlier is called: `libc6-dev`. Using `apt search` helps finding the specific package name.

Once the configure is completed successfully, the next step is to compile the source code (please

check that there are no errors reported when compiling):

```
$ make
```

In general, for these two first steps we do not need special permissions, so it is recommended that you do them within an account other than root. Instead, the last step consists of placing the binaries and other files (configuration data, libraries, ...) into the final location where we want them installed. This usually requires root permissions.

```
$ sudo make install
```

Cerify that everything is installed correctly by running the command `asosh`.

During the compilation process several temporary files should have been generated (e.g., object files). So, once the installation is completed, it is a good idea to delete these files. The Makefile usually contains the rules needed to do that easily.

What command do you use to delete temporary files?

Moreover, usually the Makefile also incorporates rules to undo all the steps made in the process of installation.

Which argument can be supplied to do this?

Bibliography

[1] “Metapackages in linux.” [Online]. Available: https://www.linux.com/training-tutorials/what-are-linux-meta-packages/