# Operating systems. Lecture 13

Michał Goliński

2019-01-08

## Introduction

### Recall

- Classical problems of multithreaded programming
  - Producer and consumer
  - Sleeping barber
  - Readers and writers
  - Dining philosophers
- Deadlocks

### Plan for today

- Linux distiributions – differences and similarities
- Administering a Linux system
  - Package manager
  - Managing services
  - Networking basics
- Overview of the most important tools

## Linux distributions

### Overview

A Linux distribution is a complete operating system based around the Linux kernel. Distribution differ in their goals, range and age of available software (and package manager) and support length.

There are over 250 active distributions. We will talk about the following groups (groups are not necessarily disjoint):

- for beginners
- for workstations
- for servers

## Distributions for beginners

- **Ubuntu** is a desktop/workstation distribution that tries to work for beginners as well as for more advanced users. It is inspired by Debian, but releases are much more frequent. It is released twice a year and tries to find a balance between stable/outdated and bleeding edge software. Users can choose between different desktop environments, but this is usually done by installing Kubuntu, Lubuntu etc.

## Distributions for beginners cont.

- **Linux Mint** is based on Ubuntu but uses a different release cadence and a different default desktop environment. Mint is more lenient on non-free software in its repositories.

## Distributions for workstations

- **Fedora** is a distribution that puts emphasis on free and bleeding edge software. Fedora aims for a release every six months (often delayed), supports many desktop environments. Fedora is supported by Red Hat and some of its features get incorporated into Red Hat Enterprise Linux.
- **OpenSUSE** is a German distribution. SUSE has a rolling release version (Tumbleweed) and fixed versions (Leap) that is released usually once a year. There is also a paid enterprise version.

## Distributions for workstations cont.

- **Arch Linux** is a rolling release distribution aimed at advanced users. Software is generally in bleeding edge versions without any stability guarantees. Has a very good and thorough wiki.

## Distributions for servers

- **Debian** had a very slow release cycle (once every two years). It usually prefers stability over bleeding edge of software. Each version has about five years of official support, but receives security updates even longer than that.
- **Red Hat Enterprise Linux** is a paid distribution made by Red Hat with a slow release cycle (lately once every three years). Each version has 10 years of official support, but this can be extended even further.

## Distributions for servers cont.

- **CentOS** is a free distribution built by community from the same source code as RHEL. It is almost exactly binary compatible with RHEL (minus the branding).

## Family tree

a family tree of current distributions can be found here (huge file).

# Package management

## Installing software form source

Assume that we want to install a program that was written in C. Historically the most often used build system was GNU Autotools, under which the program could be compiled and installed by running `./configure`, `make` and `make install`. Unfortunately there is no easy way to later uninstall the program or to make sure that no files belonging to other programs are not overwritten. Additionally, uninstalling or upgrading a library may cause programs depending on the library to fail.

## Package manager

Most software used in a Linux distribution is free (or at least can be freely distributed). This allows to have large repositories of precompiled programs that are maintained by the distribution creators. Packages contain only files and, optionally, small scripts that are run when a package is installed. They do not contain and installer. Installing itself (and uninstalling) is done by the distribution's *package manager*.

## Tasks

Package manager has many important tasks:

- makes sure all dependencies for a package are installed (e.g., libraries)
- makes sure there are no conflicts between packages (e.g., different files with the same name in two packages)
- resolves dependencies by downloading additional packages from an online repository
- upgrades installed packages to new versions

## Low and high level package manager

In most distributions these tasks are divided between two programs:

- low level package manager – works with individual package (files), physically installs packages, checks dependencies and conflicts, is able to verify if the package contents has not changed
- high level package manager – works with whole repositories, downloads packages, resolves dependencies, upgrades packages to new versions

## Low level managers

Two most often used low-level package managers are:

- `rpm` – used by Red Hat, CentOS, Fedora and SUSE.
- `dpkg` – used by Debian and Ubuntu.

Individual options are very different, but the general workflow is similar. Both maintain a database of installed packages and use package *files* as arguments for installation or package *names* for uninstallation.

### High level package managers

High level package managers are:

- `apt` – originated in Debian, used also by Ubuntu and derivatives.
- `dnf` – used by Fedora and Red Hat.
- `zypper` – used by SUSE.

All these programs can download from remote servers a database of available packages and can resolve dependencies when installing and installing them. All give also easy ways to search for software in repositories.

### `pacman`

Not all package managers use this architecture, e.g., the Arch Linux distribution uses the `pacman` package manager which combines both low and high level functionalities in a single program.

### Graphical utilities

There are also graphical tool that allow for package management. The are usually just an interface to the command line tools.

### Maintainers

Because the software is not obtained directly from *upstream* projects (e.g., LibreOffice, Mozilla), but through repositories a lot of responsibility rests with package maintainers, who create and update the packages. Usually a maintainer is responsible for many packages, closely follows the development of the upstream project and updates the package as newer version become available.

## Using a package manager in Debian/Ubuntu

### Updating the package database and packages

To update the information about available packages and their versions use:

```
# apt update
```

To update installed packages to the newest available versions, use:

```
# apt full-upgrade
```

### Searching for packages

To search the database for keywords one uses the command:

```
$ apt search keyword1 keyword2
```

This gives a list of packages where name or description contain all the given keywords. To search for packages containing a file one can use the web interface or the `apt-file` utility.

### Installing a package

To install a package with a known name one executes:

```
# apt install package-name
```

### Uninstalling a package

To uninstall a package with a known name one executes:

```
# apt remove package-name
```

This leaves configuration files around, if one wishes to completely erase everything, then `apt purge` is needed.

### Info about package

To show information about a package, use:

```
# apt show package-name
```

### Listing/verifying contents of an installed package

To list contents of an installed package, we need the low level package manager:

```
$ dpkg-query -L package-name
```

To verify if files have not been changed use:

```
dpkg --verify package-name
```

### Note

In fact there are multiple utilities: `apt-get`, `apt-cache`, `apt-config`. The command `apt` is just a simple interface to the most commonly used functions of these programs. It is still partially a work in progress.

## Distribution differences

### Package names and package splitting

It should be obvious now that different distribution use different package managers. But in different distributions the same program can be packaged under different names or the packages may be split differently.

### Package names and package splitting cont.

E.g., in Arch we have the `libreoffice-fresh` and `libreoffice-still` packages (older and newer LibreOffice, only one can be installed a time, both contain the whole office suite). In Fedora the suite is split into many individual packages: `libreoffice` (common files), `libreoffice-writer`, `libreoffice-calc` etc.

### Package names and package splitting cont.

Similarly, under Arch a library package, `zlib` say, contains the static and dynamic libraries as well as corresponding header files for developers. In Fedora this is split into `zlib` (shared library), `zlib-static` (static library), `zlib-devel` (header files). Moreover Fedora has files necessary for debugging available (`zlib-debuginfo` and `zlib-debugsource`), which is not available under Arch.

### Packages are not compatible between distributions

It should be obvious that if one distribution uses `dpkg` and another uses `rpm`, then packages are not compatible between them. But this runs deeper: each distribution uses different versions of libraries, different default configurations. This means that even packages between two `rpm`-based distributions are usually not compatible.

### Software stability

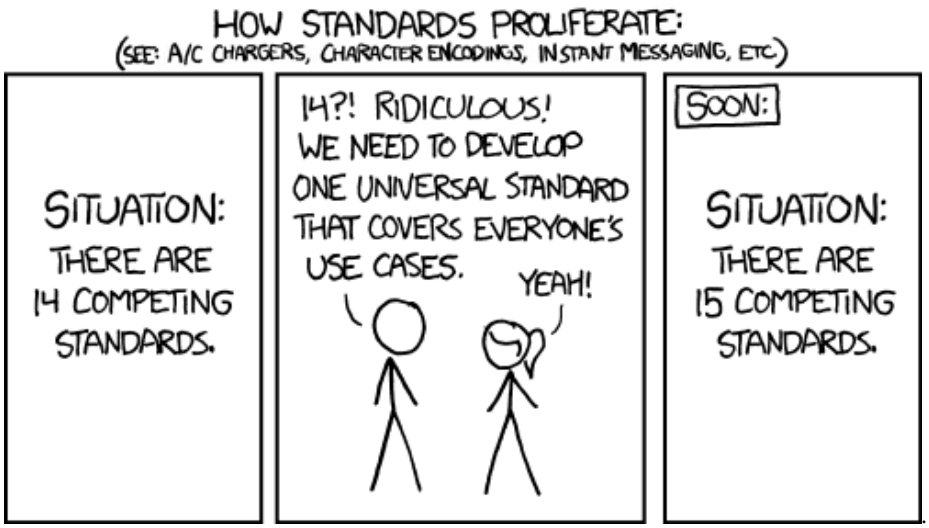
The most important difference between different distributions is the stability of software. Most distributions do not change the version of a package in a released distribution version. This gives the users stability: software they write is more-or-less guaranteed to work when updating other packages (e.g., the Python version will not be changed, as this may introduce some subtle incompatibilities). At the same time the security changes from newer versions will be backported to older versions. This is usually done by the package maintainers, sometimes together with upstream projects.

### Proprietary programs

Distributing a proprietary (i.e., closed source, limited distribution) program can be a challenging task because of the fragmentation and differing library versions. A simple solution is to create statically compiled programs or bundle the correct library versions together with the program. A more complicated solution is to maintain packages for multiple distributions. If the software is freely distributable (or at least popular), then the packages are usually created by the distribution users themselves.

### Universal packages

To fight the fragmentation at least two universal packaging formats have been introduced: Snap and Flatpak. In principle one creates, e.g., a Flatpak package that can be installed and run on multiple distributions. In practice we unfortunately have situation from xkcd927:




## Managing services

### systemd as init system

Recall that when booting the kernel runs the `init` program with PID 1, whose task is to run all the programs and services the running system needs. Today this is most often done by `systemd` – a particular `init` implementation. Introducing `systemd` used to be controversial in many distributions, but gives a much needed a level of uniformity and now managing services look similar in different distributions.

### Unit files

A service (e.g., a HTTP server) is described in so-called *unit files* (which are simple INI files). When a unit file is being started, `systemd` makes sure that runtime dependencies

(i.e., services required by the started service) are also being run. When any service dies (e.g., because of a segmentation fault), it will be restarted if the unit file requests that.

Unit files are novadays often written by upstream projects. In the past the maintainers would write startup scripts, often different in different distributions.

### Unit file example

The unit file that runs the SSH server (`sshd.service`) looks like this:

```
[Unit]
Description=OpenSSH Daemon
Wants=sshdgenkeys.service
After=sshdgenkeys.service
After=network.target

[Service]
ExecStart=/usr/bin/sshd -D
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=always

[Install]
WantedBy=multi-user.target

# This service file runs an SSH daemon that forks for each
 ↪  incoming connection.
# If you prefer to spawn on-demand daemons, use sshd.socket and
 ↪  sshd@.service.
```

### Another example

The previous unit file references the following unit file that creates the host SSH keys if necessary:

```
Description=SSH Key Generation
ConditionPathExists=|!/etc/ssh/ssh_host_dsa_key
ConditionPathExists=|!/etc/ssh/ssh_host_dsa_key.pub
ConditionPathExists=|!/etc/ssh/ssh_host_ecdsa_key
ConditionPathExists=|!/etc/ssh/ssh_host_ecdsa_key.pub
ConditionPathExists=|!/etc/ssh/ssh_host_ed25519_key
ConditionPathExists=|!/etc/ssh/ssh_host_ed25519_key.pub
ConditionPathExists=|!/etc/ssh/ssh_host_rsa_key
ConditionPathExists=|!/etc/ssh/ssh_host_rsa_key.pub

[Service]
```

```
ExecStart=/usr/bin/ssh-keygen -A
Type=oneshot
RemainAfterExit=yes
```

### Running a service

To run an installed service with the unit file named `service-name.service` one uses the command:

```
# systemctl start service-name
```

To stop a running service we use:

```
# systemctl stop service-name
```

All these commands just run the necessary commands from unit files. It there is a bug they may fail to start/stop the service.

### Starting a service at boot

To run a service at boot we need the `enable` command:

```
# systemctl enable service-name
```

To prevent a service from running at boot time:

```
# systemctl disable service-name
```

## Logs

### Kernel log

The log of the kernel is printed by the `dmesg` command. This log will give us information on hardware events, e.g., newly connected USB dongles etc. This log starts anew with each reboot.

### `systemd` log

The log for systemd and the unit files run by it can be printed by the command `journactl`. It is pretty extensive and is usually bounded by the disk size only (the log survives reboots). To see the latest information run:

```
# journalctl -xe
```

To see the log for a given service, run:

```
# journalctl -u service-name
```

### X server log

To debug problems with the X server (i.e., the window system), file `/var/log/Xorg.0.log`.

## Hardware information

### `lspci`/`lsusb`

The `lspci` command gives a list of PCI devices connected to the system. Most of them will have drivers assigned automatically by the kernel, but sometimes the information printed by `lspci` will help to hunt down a driver for an unknown device.

Similarly `lsusb` can be used to get information about USB devices.

### `dmidecode`

`dmidecode` can display information from the computer DMI table. It gives much information about hardware and firmware.

## Networking

### Overview

All network cards (ethernet, Wi-Fi, LTE etc.) are represented in the system by so-called network interfaces. The command to manage network interfaces manually is `ip`.

To communicate with a computer we generally need to assign it an IP address and set up the default route. This is usually done automatically by DHCP, but sometimes needs to be done manually.

### Getting information

To get link layer of network layer information about all the interfaces, use the following commands:

```
ip link
ip address
```

### Setting the MAC address

To set the MAC (hardware) address use the following commands (change `interface` to the right interface name):

```
# ip link set dev interface down
# ip link set dev interface address 01:23:45:56:78:90:ab
# ip link set dev interface up
```

### Setting the IP address

To set the IP address use the following command (substituting the right IP address in the CIDR notation):

```
# ip addr add 192.168.1.1/24 dev interface
```

An interface may have multiple IP addresses, to remove an address use `ip del`.

### Setting up routes

To add (delete) a route use `ip route`. For simple computers we usually only need to add the default route (gateway):

```
# ip route add default via 192.168.1.1
```

### Nameservers

The nameservers (DNS) addresses are kept in `/etc/resolv.conf`.

### DHCP client

An IP address can be assigned automatically by a DHCP server running on one of the hosts of the network. To run a DHCP client manually **one** of these commands

```
# dhcpcd interface-name
# dhclient interface-name
```

### NetworkManager

When running on a desktop computer, particularly a notebook, internet connection details tend to change (different WI-Fi networks etc.). NetworkManager is a service that keeps

note of available networks, shows a list of Wi-Fi/broadband network to users, changes connection and saves/supplies credentials when necessary.

## Important tools for administrators

### `df`

The `df` command gives an overview of free space on the mounted filesystems.

### `top`/`htop`

`top` is the standard tool for showing the running processes. It gives a table, shows CPU and memory usage, allows to kill a process if requested. `htop` is a new, more colorful version.

### `iotop`/`powertop`

`iotop` shows the current I/O bandwidth of processes.

`powertop` tries to show the power impact of processes. This might be useful when tuning the power saving when running on a laptop battery.

### SSH server

An SSH server allows to login remotely via a secure, encrypted connection. The most often used implementation of an SSH server is a part of the OpenSSH project. To run the SSH server one should start the `sshd` service. The server is configured by the `/etc/sshd.conf` file.

### HTTP server

Arguably the most often used service is the HTTP server. The traditional server is the Apache HTTP Server (`httpd`), but `nginx` is newer and gains popularity fast.