# Linux for Owners

https://goo.gl/auQmRB

v.1.1 (2019-01-29)

## Agenda

## Owning a Linux system

What do we mean by that?

## Owning a Linux system

### What do we mean by that?

**Administrative privileges** vs a normal account only.

## Owning a Linux system

### What do we mean by that?

**Administrative privileges** vs a normal account only.

Symptoms:

- access to the `root` user, or
- access to `sudo` (or `su`) commands
- having installed the system yourself (optional)
- physical access to the machine (optional)

## Owning a Linux system

### What do we mean by that?

**Administrative privileges** vs a normal account only.

Symptoms:

- access to the `root` user, or
- access to `sudo` (or `su`) commands
- having installed the system yourself (optional)
- physical access to the machine (optional)

Having admin access usually means you are responsible for **maintenance and configuration** of the system.

For that, you need a deeper knowledge of how it works.

## Typical ownership scenarios

## Typical ownership scenarios

- Linux installed as main OS on a desktop or laptop machine
  - Scenarios: work requirement, curiosity, alternative OS
  - Typically means a graphical environment

## Typical ownership scenarios

- Linux installed as main OS on a desktop or laptop machine
    - Scenarios: work requirement, curiosity, alternative OS
    - Typically means a graphical environment
- Linux used as a virtual machine guest
    - Scenarios: experimentation, sandbox, development
    - Safe way to try OSes and configurations

## Typical ownership scenarios

- Linux installed as main OS on a desktop or laptop machine
    - Scenarios: work requirement, curiosity, alternative OS
    - Typically means a graphical environment
- Linux used as a virtual machine guest
    - Scenarios: experimentation, sandbox, development
    - Safe way to try OSes and configurations
- Linux used as a server
    - Can be bare-metal or virtualized
    - Typically means a CLI environment (over SSH)

## Typical ownership scenarios

- Linux installed as main OS on a desktop or laptop machine
    - Scenarios: work requirement, curiosity, alternative OS
    - Typically means a graphical environment
- Linux used as a virtual machine guest
    - Scenarios: experimentation, sandbox, development
    - Safe way to try OSes and configurations
- Linux used as a server
    - Can be bare-metal or virtualized
    - Typically means a CLI environment (over SSH)
- Linux appliance / IoT
- Android (which you usually don't own!)

## Typical responsibilities

## Typical responsibilities

- Installing new software as needed
- Configuring software and access to hardware
- Updating software & OS
- Securing the system and installed software
- Solving problems as they appear

## Course expectations

This is a crash-course limited to one day.

- You are expected to know the basics of working with Linux. At a minimum, working with command line and editing files from it.
- You were asked to prepare a VM with Linux, that way you can safely try admin commands and reboot the OS without affecting your main system.
- Some commands can differ per Linux "flavour". The course sticks to Ubuntu / Debian, and may need to be adapted to other distributions.
- The course is intended to be an introduction; to dig deeper into topics you'll need to do your own research.

## Linux landscape: What's a distribution?

"Linux" is the name of the OS kernel, and needs a lot of other software to be a functional operating system.

One can, in principle, assemble this collection of software themselves: such an approach is called Linux From Scratch.

## Linux landscape: What's a distribution?

"Linux" is the name of the OS kernel, and needs a lot of other software to be a functional operating system.

One can, in principle, assemble this collection of software themselves: such an approach is called Linux From Scratch.

It's long, requires in-depth knowledge of the system and most importantly leaves keeping the system up to date to the user.

A better approach is to use a **distribution**, which offers:

- A curated collection of software tested to work together.
- A **package manager** to automate software install tasks.
- Pre-compiled software for faster installation.
- Security updates and fixes for critical bugs.
- Varying balance between stability and faster updates.
- *Optional:* paid support.

## Linux landscape: Popular distributions

There are many families of distributions, sharing package managers and maybe parts of the package collection.

## Linux landscape: Popular distributions

There are many families of distributions, sharing package managers and maybe parts of the package collection.

- Debian (package manager: `apt` / .deb packages)
  - Ubuntu
    - Kubuntu, Lubuntu, …
    - Linux Mint
  - Kali, Tails, Devuan

# Linux landscape: Popular distributions

There are many families of distributions, sharing package managers and maybe parts of the package collection.

- Debian (package manager: `apt` / .deb packages)
  - Ubuntu
    - Kubuntu, Lubuntu, …
    - Linux Mint
  - Kali, Tails, Devuan
- RHEL (package manager: `yum` / .rpm packages)
  - Fedora
  - CentOS, Scientific Linux
  - SUSE, OpenSUSE

## Linux landscape: Popular distributions

There are many families of distributions, sharing package managers and maybe parts of the package collection.

- Debian (package manager: `apt` / .deb packages)
    - Ubuntu
        - Kubuntu, Lubuntu, …
        - Linux Mint
    - Kali, Tails, Devuan
- RHEL (package manager: `yum` / .rpm packages)
    - Fedora
    - CentOS, Scientific Linux
    - SUSE, OpenSUSE
- Arch Linux (package manager: `pacman`)
    - Manjaro Linux
- Gentoo (package manager: Portage / `emerge`)
    - Chrome OS
- Alpine Linux
- …

## Linux landscape: Rolling / Static / LTS

Various Linux distributions take different approaches to software updates.

Programs update at their own pace, called "upstream".

## Linux landscape: Rolling / Static / LTS

Various Linux distributions take different approaches to software updates.

Programs update at their own pace, called "upstream".

**Rolling distributions** try to keep up with upstream version updates, with minimal changes to keep it compatible with the rest of the software stack.

## Linux landscape: Rolling / Static / LTS

Various Linux distributions take different approaches to software updates.

Programs update at their own pace, called "upstream".

**Rolling distributions** try to keep up with upstream version updates, with minimal changes to keep it compatible with the rest of the software stack.

**Static distributions** lock down software versions at the time of release, in order to provide more testing and stability.

## Linux landscape: Rolling / Static / LTS

Various Linux distributions take different approaches to software updates.

Programs update at their own pace, called "upstream".

**Rolling distributions** try to keep up with upstream version updates, with minimal changes to keep it compatible with the rest of the software stack.

**Static distributions** lock down software versions at the time of release, in order to provide more testing and stability.

However, critical bugfixes and security patches have to be ported back "downstream" to those "frozen" versions.

Some static distributions have **Long Term Support** releases that guarantee longer availability of such patches.

## Assuming superpowers

How do administrative powers work in Linux?

Compared to regular users, the user with ID 0, or `root`, is treated as **superuser**: it has the ability to bypass all usual access restrictions.

## Assuming superpowers

How do administrative powers work in Linux?

Compared to regular users, the user with ID 0, or **`root`**, is treated as **superuser**: it has the ability to bypass all usual access restrictions.

In some Linux distributions a password for `root` is set at install time and can be used to log in directly.

However, most of the time users with administrative privileges will have access to use **`sudo` command**.

*Mnemonic:*

`sudo` stands for "**su**peruser **do**", because it executes some other command as superuser instead of current user.

## Assuming superpowers: `sudo`

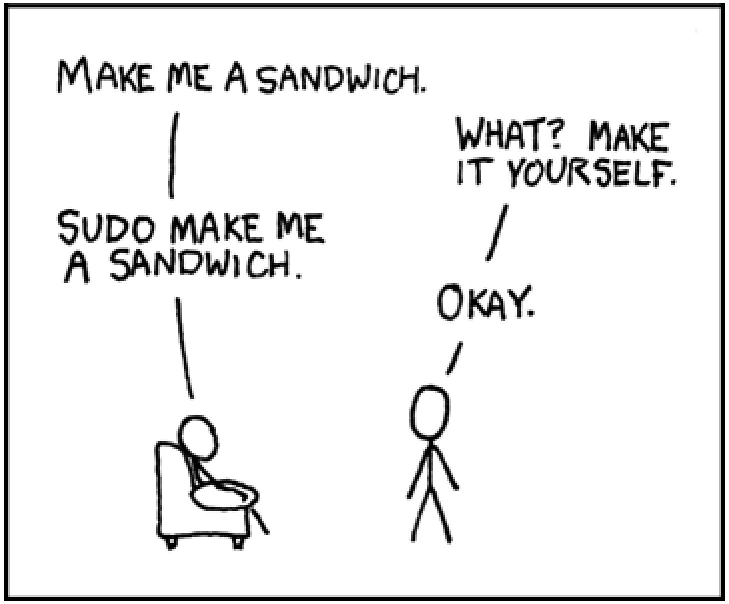


*Image credit:* [XKCD #149]

## Assuming superpowers: `sudo`

`sudo [command]` executes the command as `root` instead of the current user:

```
owner@linux:~$ whoami
owner
owner@linux:~$ sudo whoami
[sudo] password for owner:
```

## Assuming superpowers: `sudo`

`sudo [command]` executes the command as `root` instead of the current user:

```
owner@linux:~$ whoami
owner
owner@linux:~$ sudo whoami
[sudo] password for owner:
```

Normally, `sudo` asks for authentication: it requires you to supply **your own** password (not the `root` password!).

```
owner@linux:~$ sudo whoami
[sudo] password for owner: <type in owner's password>
root
owner@linux:~$
```

## Assuming superpowers: `sudo`

`sudo` remembers that you have authenticated, and subsequent commands do not require password for 15 minutes:

```
owner@linux:~$ sudo whoami
root
owner@linux:~$
```

## Assuming superpowers: `sudo`

`sudo` remembers that you have authenticated, and subsequent commands do not require password for 15 minutes:

```
owner@linux:~$ sudo whoami
root
owner@linux:~$
```

If you need to do a lot of stuff as root, you can use `sudo -i` to start a login shell as `root`:

```
owner@linux:~$ sudo -i
root@linux:~# whoami
root
root@linux:~#
```

*Mnemonic:*

`-i` is short for **interactive**

*Reference:* `man sudo_root`

## To root or not to root

Forgetting `sudo` before a command that requires it can be jarring. Therefore, there is great temptation to use a `root` shell when mostly admin tasks are involved.

## To root or not to root

Forgetting `sudo` before a command that requires it can be jarring. Therefore, there is great temptation to use a `root` shell when mostly admin tasks are involved.

Problem is, there are no "handrails" on what the `root` user can do. It's far too easy to shoot oneself in the foot.

An error in a program run under `root` can break critical parts of the system, which won't happen under a regular user.

## To root or not to root

Forgetting `sudo` before a command that requires it can be jarring. Therefore, there is great temptation to use a `root` shell when mostly admin tasks are involved.

Problem is, there are no "handrails" on what the `root` user can do. It's far too easy to shoot oneself in the foot.

An error in a program run under `root` can break critical parts of the system, which won't happen under a regular user.

From a security perspective, running complex, or untrusted, programs under `root` opens you up for malicious actions as well.

## To root or not to root: a horror story

Running this command as a `root` is a classic mean prank:

```
root@linux:~# rm -rf /
rm: it is dangerous to operate recursively on '/'
rm: use --no-preserve-root to override this failsafe
```

Fortunately, as written it doesn't work, there's a failsafe.

## To root or not to root: a horror story

Running this command as a `root` is a classic mean prank:

```
root@linux:~# rm -rf /
rm: it is dangerous to operate recursively on '/'
rm: use --no-preserve-root to override this failsafe
```

Fortunately, as written it doesn't work, there's a failsafe.

However, consider this line:

```
(DO NOT RUN THIS)
root@linux:~# rm -rf $FOLDER/*
```

This is supposed to delete everything in folder `$FOLDER`.

But in case that variable is unset, it will happily destroy everything on the system — if run as `root`.

## To root or not to root

In addition, if files are created while running as `root` it usually leads to them being inaccessible by normal users.

This can lead to errors when running the same commands later as a normal user.

## To root or not to root

In addition, if files are created while running as `root` it usually leads to them being inaccessible by normal users.

This can lead to errors when running the same commands later as a normal user.

Most workflows are a mixture of commands that require superuser and those that don't.

It's good practice to be aware of it and run only those parts that need it with `sudo`.

## To root or not to root

In addition, if files are created while running as `root` it usually leads to them being inaccessible by normal users.

This can lead to errors when running the same commands later as a normal user.

Most workflows are a mixture of commands that require superuser and those that don't.

It's good practice to be aware of it and run only those parts that need it with `sudo`.

It's not a hard rule. Running a set of commands as `root` has its place, but one needs to be careful and mindful of the above nuances.

## root and SSH

By default, even if you set a password for the `root` user, you won't be able to connect via SSH with this password.

This is done to discourage brute-forcing the `root` password.

## root and SSH

By default, even if you set a password for the `root` user, you won't be able to connect via SSH with this password.

This is done to discourage brute-forcing the `root` password.

This encourages people to connect as their user and elevate with `sudo` if necessary. This creates more meaningful logs.

Also, if your SSH key is compromised but your user password is not, the attacker won't be able to elevate to `root` even if they can connect.

## root and SSH

By default, even if you set a password for the `root` user, you won't be able to connect via SSH with this password.

This is done to discourage brute-forcing the `root` password.

This encourages people to connect as their user and elevate with `sudo` if necessary. This creates more meaningful logs.

Also, if your SSH key is compromised but your user password is not, the attacker won't be able to elevate to `root` even if they can connect.

It's still possible to connect directly as `root` by weakening the security (bad idea) or adding an SSH key to the `root` account. However, previous warnings apply.

## How is `sudo` configured?

What enables certain users to use `sudo`?

By default, members of a specific group (e.g. `sudo`, `admin`, `wheel`) are allowed all `root` privileges after verifying their password.

## How is `sudo` configured?

What enables certain users to use `sudo`?

By default, members of a specific group (e.g. `sudo`, `admin`, `wheel`) are allowed all `root` privileges after verifying their password.

There's a `/etc/sudoers` file describing the system policy.

It can be fine-tuned to allow some users access only to specific commands, or dropping the password requirement.

*Advice:*

When editing `sudoers` file, always use the `visudo` script. It sanity-checks changes so that you don't lock yourself out.

*References:* `man sudoers`, [Understanding the sudoers file]

## Exercise: inspect the `sudoers` file

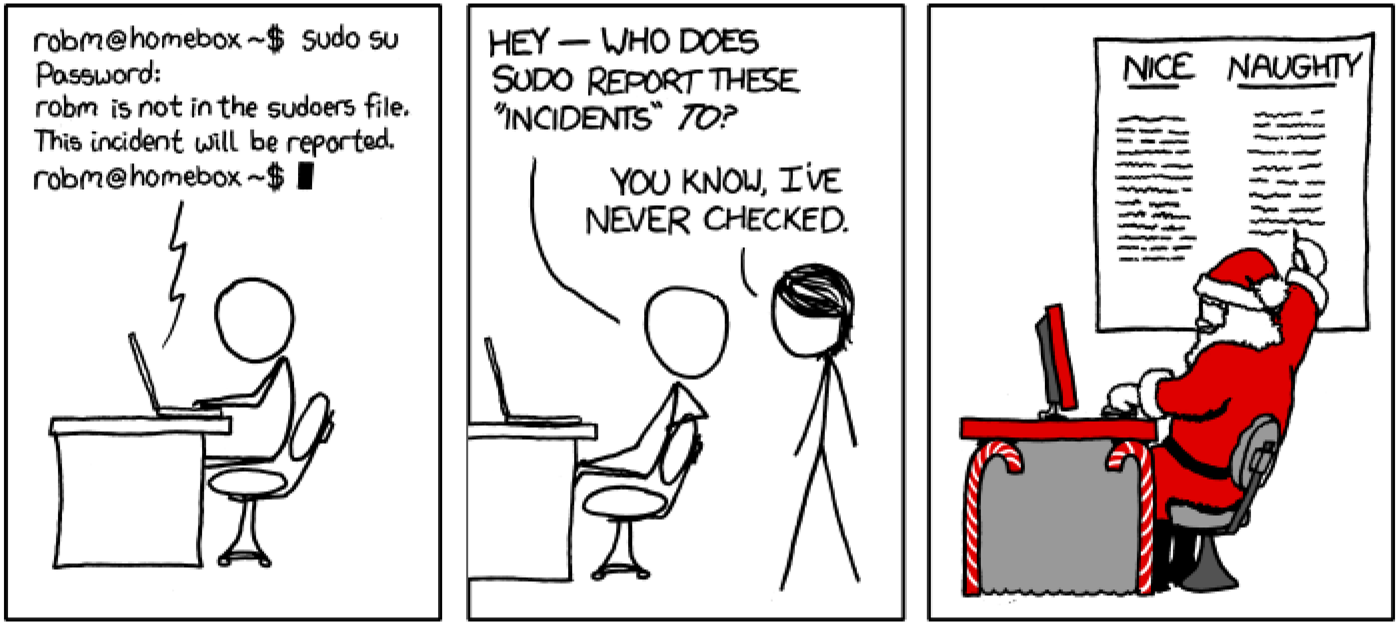
*Exercise:*

1. Try examining the contents of `/etc/sudoers` file:

```
owner@linux:~$ less /etc/sudoers
```

2. Explain the problem
3. Fix the problem (using `sudo`)

## Bonus: violating the `sudoers` policy



```
robm@homebox ~$ sudo su
Password:
robm is not in the sudoers file.
This incident will be reported.
robm@homebox ~$ █
```

HEY — WHO DOES SUDO REPORT THESE "INCIDENTS" TO?
YOU KNOW, I'VE NEVER CHECKED.
NICE
NAUGHTY

*Image credit:* [XKCD #838]

## Linux filesystem map: what goes where?

Before we proceed, it's important to understand common filesystem locations in Linux.

It differs a little from distribution to distribution, but generally systems adhere to the **Filesystem Hierarcy Standard**, or FHS (3.0 currently).

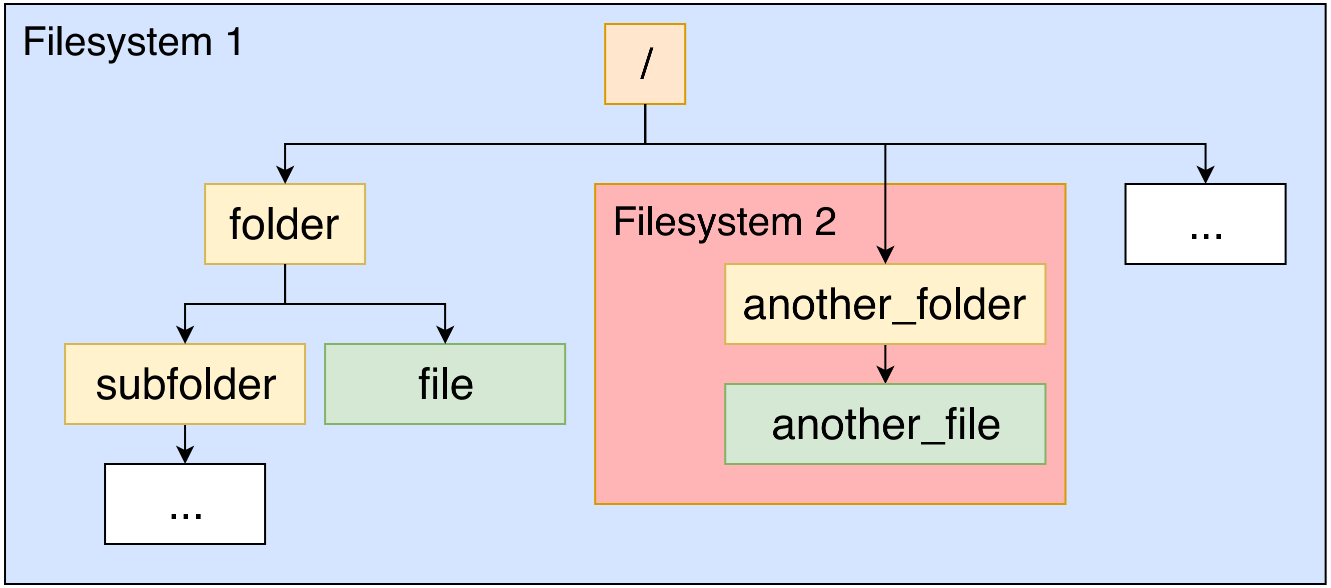
Not every entry in the standard will be explained, but "points of interest" will be shown.

*References:* Wikipedia entry on FHS, FHS 3.0 specification

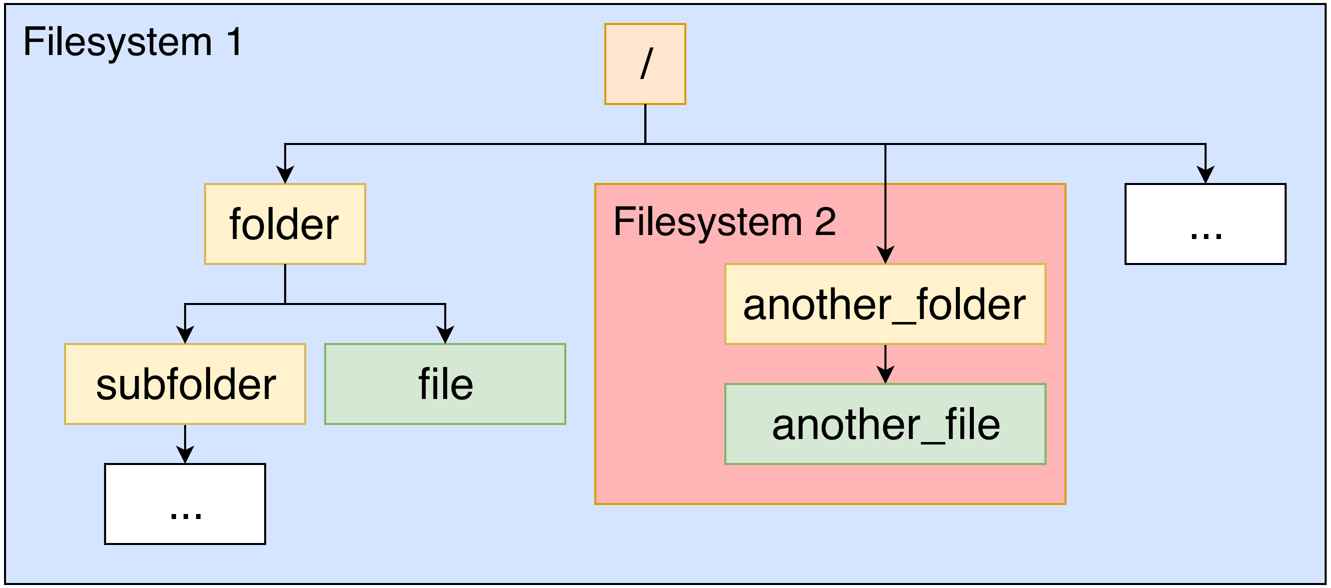
# Linux filesystem: a common tree

Under UNIX conventions, there's only a single filesystem tree visible in the system.

There can be multiple disks, or multiple partitions on those disks, each with its own filesystem trees, but to be usable they are inserted in the common file tree as subtrees:



# Linux filesystem: a common tree



This process is called **mounting**. You can examine currently mounted filesystems with the `mount` command:

```
owner@linux:~$ mount
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
[...]
```

## Linux filesystem: /home sweet home

As a user, you're probably familiar with `/home`.

Home directories for each user are usually stored there:

```
owner@linux:~$ echo $HOME
/home/owner
owner@linux:~$ ls -l /home/
total 6
drwxr-xr-x 4 owner  owner  4096 Jan 14 00:03 owner
drwxr-xr-x 2 user   user   4096 Jan 15 18:31 user
```

The `root` user is an exception:

```
owner@linux:~$ sudo -i
root@linux:~# echo $HOME
/root
```

This is done to help with troubleshooting, if `/home` is a separate filesystem and something goes wrong with it.

## Linux filesystem: configuration, *et cetera*

We have already saw **`/etc`** mentioned when we looked at `sudoers` policy.

In fact, by convention all **system-wide configuration** files are stored there, often grouped into folders by topic/program.

Most files are only writable by `root`, and some highly-sensitive ones, like `/etc/sudoers`, are not readable by anyone else.

*Note:*

User-specific configuration (and persistent state) is usually stored in the user's home folder as hidden files/folders, e.g. `/home/owner/.bashrc`

```
owner@linux:~$ ls -a ~
.  ..  .bash_history  .bash_logout  .bashrc  .cache  .gnupg  .local  .ssh ...
```

## Linux filesystem: where software lives

Executable (**"binary"**) files for software installed system-wide is read-only for regular users and is organized in (up to) 3 tiers of hierarchy of **`sbin`/`bin`** folders:

- `/sbin`, `/bin` - essential system software

## Linux filesystem: where software lives

Executable (**"binary"**) files for software installed system-wide is read-only for regular users and is organized in (up to) 3 tiers of hierarchy of **`sbin/bin`** folders:

- **`/`**`sbin`, **`/`**`bin` - essential system software
- **`/usr/`**`sbin`, **`/usr/`**`bin` - more software that's part of the OS

## Linux filesystem: where software lives

Executable (**"binary"**) files for software installed system-wide is read-only for regular users and is organized in (up to) 3 tiers of hierarchy of `sbin/bin` folders:

- `/sbin`, `/bin` - essential system software
- `/usr/sbin`, `/usr/bin` - more software that's part of the OS
- `/usr/local/sbin`, `/usr/local/bin` - additional software

## Linux filesystem: where software lives

Executable (**"binary"**) files for software installed system-wide is read-only for regular users and is organized in (up to) 3 tiers of hierarchy of **`sbin`/`bin`** folders:

- **`/`**`sbin`, **`/`**`bin` - essential system software
- **`/usr/`**`sbin`, **`/usr/`**`bin` - more software that's part of the OS
- **`/usr/local/`**`sbin`, **`/usr/local/`**`bin` - additional software

These locations can have `lib` and `include` folders for library and header files, and `share` folders for program data.

## Linux filesystem: where software lives

Executable (**"binary"**) files for software installed system-wide is read-only for regular users and is organized in (up to) 3 tiers of hierarchy of **`sbin`/`bin`** folders:

- **`/`**`sbin`, **`/`**`bin` - essential system software
- **`/usr/`**`sbin`, **`/usr/`**`bin` - more software that's part of the OS
- **`/usr/local/`**`sbin`, **`/usr/local/`**`bin` - additional software

These locations can have `lib` and `include` folders for library and header files, and `share` folders for program data.

Finally, **`/opt`** provides a location for installing extra software without interfering with `/usr`.

## Linux filesystem: where software lives

Executable (**"binary"**) files for software installed system-wide is read-only for regular users and is organized in (up to) 3 tiers of hierarchy of `sbin/bin` folders:

- `/sbin`, `/bin` - essential system software
- `/usr/sbin`, `/usr/bin` - more software that's part of the OS
- `/usr/local/sbin`, `/usr/local/bin` - additional software

These locations can have `lib` and `include` folders for library and header files, and `share` folders for program data.

Finally, `/opt` provides a location for installing extra software without interfering with `/usr`.

In practice, the difference between these locations is not very well defined. Some distributions don't differentiate at all.

## Linux filesystem: not-quite-files

Several locations on the filesystem contain virtual "files" that can be read and/or written to, but represent something else.

## Linux filesystem: not-quite-files

Several locations on the filesystem contain virtual "files" that can be read and/or written to, but represent something else.

- `/dev` contains **device** files that serve as interfaces to hardware that the kernel sees and has drivers for;

## Linux filesystem: not-quite-files

Several locations on the filesystem contain virtual "files" that can be read and/or written to, but represent something else.

- `/dev` contains **device** files that serve as interfaces to hardware that the kernel sees and has drivers for;
- `/proc` contains information and interfaces to control running **processes** and some kernel systems;

## Linux filesystem: not-quite-files

Several locations on the filesystem contain virtual "files" that can be read and/or written to, but represent something else.

- **`/dev`** contains **device** files that serve as interfaces to hardware that the kernel sees and has drivers for;
- **`/proc`** contains information and interfaces to control running **processes** and some kernel systems;
- **`/sys`** contains kernel and **system** information and interfaces to control it. It's a newer method than `/proc`.

Those locations are, again, usually read-only (or not accessible) to regular users.

## Linux filesystem: variable and temporary

Besides fixed program data, the filesystem has locations for **variable** data, like temporary files and persistent state.

# Linux filesystem: variable and temporary

Besides fixed program data, the filesystem has locations for **variable** data, like temporary files and persistent state.

- `/var` represents variable data. A couple notable folders:
    - `/var/log` for system logs
    - `/var/lib` for data modified by programs, e.g. databases

## Linux filesystem: variable and temporary

Besides fixed program data, the filesystem has locations for **variable** data, like temporary files and persistent state.

- **`/var`** represents variable data. A couple notable folders:
    - **`/var/log`** for system logs
    - **`/var/lib`** for data modified by programs, e.g. databases

- **`/run`** contains state information since startup.
- **`/tmp`** is for temporary data, and is user-writable.

Both `/run` and `/tmp` locations are emptied on boot, and in fact usually exist only in memory.

## Linux filesystem: variable and temporary

Besides fixed program data, the filesystem has locations for **variable** data, like temporary files and persistent state.

- **`/var`** represents variable data. A couple notable folders:
  - **`/var/log`** for system logs
  - **`/var/lib`** for data modified by programs, e.g. databases

- **`/run`** contains state information since startup.
- **`/tmp`** is for temporary data, and is user-writable.

Both `/run` and `/tmp` locations are emptied on boot, and in fact usually exist only in memory.

- **`/var/tmp`** is also for temporary data, but it survives reboot.

## Linux filesystem: other storage

When extra storage is connected to the system, there are conventional places to add it to the filesystem.

## Linux filesystem: other storage

When extra storage is connected to the system, there are conventional places to add it to the filesystem.

- **`/media`** is populated with filesystems from removable storage added to the system: USB drives, CD disks, etc.

  Often, such devices are added to `/media` automatically.

## Linux filesystem: other storage

When extra storage is connected to the system, there are conventional places to add it to the filesystem.

- **`/media`** is populated with filesystems from removable storage added to the system: USB drives, CD disks, etc.

  Often, such devices are added to `/media` automatically.

- **`/mnt`** is the traditional place to mount additional filesystems temporarily.

## Linux filesystem: where everything starts

Last but not least, **`/boot`** contains essential files to initialize Linux when the computer starts up.

Typical contents:

- `/boot/grub/` contains **bootloader** components
- `/boot/vmlinuz*` files are compressed Linux kernels
- `/boot/initrd*` files are initial filesystems that are used until "real" root becomes available

*Reference:* A broad overview of how modern Linux systems boot

## Linux filesystem: where everything starts

Last but not least, **`/boot`** contains essential files to initialize Linux when the computer starts up.

Typical contents:

- `/boot/grub/` contains **bootloader** components
- `/boot/vmlinuz*` files are compressed Linux kernels
- `/boot/initrd*` files are initial filesystems that are used until "real" root becomes available

*Reference:* A broad overview of how modern Linux systems boot

*Note:*

On some OS configurations, `/boot` is a small separate filesystem. Since it usually contains several versions of the kernel as a backup, in worst case it can run out of space.

In that case, you'll need to clean up older kernels before

## Linux filesystem: parts, assemble!

As mentioned, the `mount` command can be used to list currently-mounted filesystems.

## Linux filesystem: parts, assemble!

As mentioned, the `mount` command can be used to list currently-mounted filesystems.

To mount a filesystem, `mount [WHAT] [WHERE]` format is used.

Here, `[WHAT]` identifies where the filesystem can be accessed (e.g. a physical drive partition), and `[WHERE]` identifies the directory where its contents should be mapped.

## Linux filesystem: parts, assemble!

As mentioned, the **`mount`** command can be used to list currently-mounted filesystems.

To mount a filesystem, **`mount [WHAT] [WHERE]`** format is used.

Here, `[WHAT]` identifies where the filesystem can be accessed (e.g. a physical drive partition), and `[WHERE]` identifies the directory where its contents should be mapped.

To unmount a filesystem, **`umount [WHAT]`** (or `umount [WHERE]`) command can be used. Typically, only `root` can mount or unmount.

## Linux filesystem: parts, assemble!

As mentioned, the **`mount`** command can be used to list currently-mounted filesystems.

To mount a filesystem, **`mount [WHAT] [WHERE]`** format is used.

Here, `[WHAT]` identifies where the filesystem can be accessed (e.g. a physical drive partition), and `[WHERE]` identifies the directory where its contents should be mapped.

To unmount a filesystem, **`umount [WHAT]`** (or `umount [WHERE]`) command can be used. Typically, only `root` can mount or unmount.

For permanent mount rules, e.g. what gets mounted on boot, see the **`/etc/fstab`** file.

*References:* `man mount`, `man umount`, `man fstab`

## Exercise: adding a filesystem

Let's see `mount` and `fstab` in action. We'll create a file that holds a small filesystem, mount it, and then set it up to mount automatically at boot.

Let's create a 100MB file and an ext4 filesystem on it:

```
owner@linux:~$ truncate -s 100m loop_file
owner@linux:~$ mkfs.ext4 loop_file
```

Then, let's create a mount point (where the filesystem will be mapped):

```
owner@linux:~$ sudo mkdir /mnt/tiny
owner@linux:~$ ls -la /mnt/tiny/
total 8
drwxr-xr-x 2 root root 4096 Jan 16 09:50 .
drwxr-xr-x 3 root root 4096 Jan 16 09:50 ..
```

## Exercise: mounting a filesystem

Let's mount our tiny filesystem.

```
owner@linux:~$ sudo mount /home/owner/loop_file /mnt/tiny
owner@linux:~$ mount | grep tiny
/home/owner/loop_file on /mnt/tiny type ext4 (rw,relatime,data=ordered)
```

As a regular user, we can't write to it:

```
owner@linux:~$ touch /mnt/tiny/test
touch: cannot touch '/mnt/tiny/test': Permission denied
```

Let's make it writable:

```
owner@linux:~$ sudo chmod a+w /mnt/tiny/
owner@linux:~$ touch /mnt/tiny/test
owner@linux:~$ ls -la /mnt/tiny/
total 17
drwxrwxrwx 3 root  root   1024 Jan 16 10:56 ./
drwxr-xr-x 3 root  root   4096 Jan 16 09:50 ../
drwx------ 2 root  root  12288 Jan 16 09:49 lost+found/
-rw-r--r-- 1 owner owner     0 Jan 16 10:56 test
```

## Exercise: unmounting a filesystem

```
owner@linux:~$ ls -la /mnt/tiny/
total 17
drwxrwxrwx 3 root  root   1024 Jan 16 10:56 ./
drwxr-xr-x 3 root  root   4096 Jan 16 09:50 ../
drwx------ 2 root  root  12288 Jan 16 09:49 lost+found/
-rw-r--r-- 1 owner owner     0 Jan 16 10:56 test
```

Let's unmount the filesystem and examine the situation again:

```
owner@linux:~$ sudo umount /mnt/tiny/
owner@linux:~$ ls -la /mnt/tiny/
total 8
drwxr-xr-x 2 root root 4096 Jan 16 09:50 .
drwxr-xr-x 3 root root 4096 Jan 16 09:50 ..
```

Our file is not there.

Note that the mount point is not writable by all, as our `chmod` command affected the tiny filesystem's root folder and not the mount point.

## Exercise: add to `fstab`

Let's add our mount to `fstab`.

*Advice:*

You can edit configuration files with `sudo nano <FILENAME>` (or other editor), but it's better to use `sudoedit <FILENAME>`.

See sudoedit: why use it over sudo vi? for an explanation.

```
owner@linux:~$ sudoedit /etc/fstab
```

It opens a file with already one or more records present. We need to add one.

## Exercise: `fstab` format

An `fstab` record contains whitespace-separated fields:

- [file system] - where to find the filesystem
- [mount point] - where to map it in the system
- [type] - what type of filesystem it is (`-t` flag)
- [options] - various settings you can tweak (`-o` flag)
- [dump] - 0/1, setting for a very old backup solution
  **In practice, always 0**
- [pass] - order to check the filesystem for errors on boot

Let's add our filesystem, at the end of the file:

```
/home/owner/loop_file /mnt/tiny ext4 defaults 0 2
```

Take care not to modify other entries. Save and exit.

## Exercise: filesystem in `fstab`

Now that our example is in `fstab`, we can mount it easier:

```
owner@linux:~$ sudo mount /mnt/tiny/
owner@linux:~$ ls /mnt/tiny
lost+found  test
```

You can mount all filesystems in `fstab` at once with `mount -a`.

Further, our example should now be accessible after a reboot. To prevent automatic mounting at boot time, use `noauto` option.

*Note:*

To be able to mount a filesystem, corresponding drivers/utilities need to be installed. If you fail to mount something, like a network drive, check installed packages.

## Remark: mounting network filesystems

Mounting filesystems over the network works similarly.

Note that you usually need to install protocol specific packages, e.g.

- `cifs-utils` for Windows / SMB shares (`\\server\folder\`)
- `nfs-common` for UNIX-style NFS shares (`server:/path`)
- `sshfs` for mounting remote folders over SSH

## Working with the package manager

To simplify software installation, updates and removal, distributions provide a **package manager** and a collection of "official" packages.

Command examples for `apt`, Debian/Ubuntu package manager, will be shown, but principles apply to other managers.

## Packages: software sources

To install software, `apt` needs up-to-date information about packages that are available.

**`/etc/apt/sources.list`** contains the information on **repositories** to get information from for system packages.

## Packages: software sources

To install software, `apt` needs up-to-date information about packages that are available.

**`/etc/apt/sources.list`** contains the information on **repositories** to get information from for system packages.

A command needs to be run to update a local catalog of package information:

```
owner@linux:~ $ sudo apt update
[...]
Get:8 http://ch.archive.ubuntu.com/ubuntu bionic-security/universe amd64 Packag
Fetched 1798 kB in 1s (1768 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
12 packages can be upgraded. Run 'apt list --upgradable' to see them.
```

## Packages: upgrading installed packages

Use **`apt upgrade`** command to install updates `apt` is aware of.

## Packages: upgrading installed packages

Use **apt upgrade** command to install updates apt is aware of.

```
owner@linux:~$ sudo apt upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages will be upgraded:
  gjs gnome-software gnome-software-common gnome-software-plugin-snap gvfs
  gvfs-backends gvfs-bin gvfs-common gvfs-daemons gvfs-fuse gvfs-libs libgjs0g
  libsmbclient libwbclient0 linux-firmware samba-libs ubuntu-software
17 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Need to get 71.6 MB/74.6 MB of archives.
After this operation, 3'258 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

After confirmation, apt will download and apply updates.

## Packages: upgrading installed packages

Use `apt upgrade` command to install updates `apt` is aware of.

```
owner@linux:~$ sudo apt upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages will be upgraded:
  gjs gnome-software gnome-software-common gnome-software-plugin-snap gvfs
  gvfs-backends gvfs-bin gvfs-common gvfs-daemons gvfs-fuse gvfs-libs libgjs0g
  libsmbclient libwbclient0 linux-firmware samba-libs ubuntu-software
17 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Need to get 71.6 MB/74.6 MB of archives.
After this operation, 3'258 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

After confirmation, `apt` will download and apply updates.

*Note:*

Sometimes, an upgrade needs to remove some (replaced) packages. In that case, `apt dist-upgrade` is used.

## Packages: to upgrade or not to upgrade?

Static distributions like Ubuntu only update packages when serious bugfixes or security fixes are needed, with little to no functionality changes.

As such, it's usually safe to upgrade OS-provided packages.

## Packages: to upgrade or not to upgrade?

Static distributions like Ubuntu only update packages when serious bugfixes or security fixes are needed, with little to no functionality changes.

As such, it's usually safe to upgrade OS-provided packages.

If you're using a rolling distribution, or a third-party repository, they can introduce breaking changes. Be prepared and read available changelogs to determine when to upgrade.

## Packages: to upgrade or not to upgrade?

Static distributions like Ubuntu only update packages when serious bugfixes or security fixes are needed, with little to no functionality changes.

As such, it's usually safe to upgrade OS-provided packages.

If you're using a rolling distribution, or a third-party repository, they can introduce breaking changes. Be prepared and read available changelogs to determine when to upgrade.

If you need to stop a particular package from upgrading, you can put it on **hold**:

```
owner@linux:~$ sudo apt-mark hold bash
bash set on hold.
owner@linux:~$ sudo apt-mark unhold bash
Canceled hold on bash.
```

## Packages: when to upgrade?

For most updates, upgrades will not disrupt normal workflow and can be applied at any time.

Note that upgrading running services (e.g. a web server) requires them to restart, briefly becoming unavailable. You may need to plan your maintenance in advance.

## Packages: when to upgrade?

For most updates, upgrades will not disrupt normal workflow and can be applied at any time.

Note that upgrading running services (e.g. a web server) requires them to restart, briefly becoming unavailable. You may need to plan your maintenance in advance.

Some upgrades (mostly kernel updates) can't be applied without a reboot. The system will nag you about it, but you won't be forced to reboot.

## Packages: when to upgrade?

For most updates, upgrades will not disrupt normal workflow and can be applied at any time.

Note that upgrading running services (e.g. a web server) requires them to restart, briefly becoming unavailable. You may need to plan your maintenance in advance.

Some upgrades (mostly kernel updates) can't be applied without a reboot. The system will nag you about it, but you won't be forced to reboot.

General advice for any software updates: be prepared to spend some time dealing with unexpected problems.

## Packages: Finding system packages

If you're missing some software and it's available in OS repositories, it's very easy to install.

The shell can even **suggest the package** you're missing:

```
owner@linux:~$ fortune

Command 'fortune' not found, but can be installed with:

sudo apt install fortune-mod
```

Otherwise, you can **search known packages** by package name or description with **`apt search`**:

```
owner@linux:~$ apt search cow
[..long list..]
owner@linux:~$ apt search --names-only cow
[..shorter list..]
```

## Packages: Installing system packages

Once you know the name of the package(s), you can install them with **`apt install`**:

```
owner@linux:~$ sudo apt install fortune-mod cowsay
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  fortunes-min librecode0
Suggested packages:
  filters cowsay-off fortunes
The following NEW packages will be installed:
  cowsay fortune-mod fortunes-min librecode0
0 upgraded, 4 newly installed, 0 to remove and 0 not upgraded.
Need to get 638 kB of archives.
After this operation, 2'220 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

After confirmation, `apt` will download and install them.

## Bonus: just what did we just install?!

Let's test the software we just installed.

```
owner@linux:~$ fortune | cowsay

 _____________________________________
/ Truth is the most valuable thing we \
| have -- so let us economize it.     |
|                                     |
\ -- Mark Twain                       /
 -------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

*Exercise:*

This could look better: we're missing some pretty colors.

Find the package that provides the `lolcat` application, install it and pipe the output through it.

## Packages: dependency resolution

Let's look at that `apt install` output again.

```
The following additional packages will be installed:
  fortunes-min librecode0
Suggested packages:
  filters cowsay-off fortunes
The following NEW packages will be installed:
  cowsay fortune-mod fortunes-min librecode0
0 upgraded, 4 newly installed, 0 to remove and 0 not upgraded.
```

Often, packages require other packages (e.g. libraries, data files) to be installed to function. The package manager's job is to find and install everything that's needed.

*Note:*

Rarely, a package is incompatible with already installed ones. You will then be offered to remove packages to resolve this. In that case, proceed with caution.

## Packages: getting rid of them

If you no longer need a package, you can use `apt remove`:

```
owner@linux:~$ sudo apt remove fortune-mod
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required
  fortunes-min librecode0
Use 'sudo apt autoremove' to remove them.
The following packages will be REMOVED:
  fortune-mod
0 upgraded, 0 newly installed, 1 to remove and 12 not upgraded.
After this operation, 110 kB disk space will be freed.
Do you want to continue? [Y/n]
```

After confirmation, the program will be removed, but its configuration files will remain in case you reinstall.

## Packages: getting rid of them

If you no longer need a package, you can use `apt remove`:

```
owner@linux:~$ sudo apt remove fortune-mod
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required
  fortunes-min librecode0
Use 'sudo apt autoremove' to remove them.
The following packages will be REMOVED:
  fortune-mod
0 upgraded, 0 newly installed, 1 to remove and 12 not upgraded.
After this operation, 110 kB disk space will be freed.
Do you want to continue? [Y/n]
```

After confirmation, the program will be removed, but its configuration files will remain in case you reinstall.

If you want to completely clean up the package, use `apt purge`:

```
owner@linux:~$ sudo apt purge fortune-mod
```

## Packages: orphaned dependencies

Take note of these lines in `apt remove` output:

```
The following packages were automatically installed and are no longer required
  fortunes-min librecode0
```

`apt` is informing you that `fortunes-min` and `librecode0` were installed as dependencies, but **nothing requires them anymore**.

Therefore, it's highly likely that they can be removed without affecting anything. However, `apt` doesn't do that automatically.

To allow `apt` to do that, use **`apt autoremove`**:

```
owner@linux:~$ sudo apt autoremove
[...]
```

## Packages: getting information

If you want to know more about a package, you can query for information.

## Packages: getting information

If you want to know more about a package, you can query for information.

`apt show` displays metadata about a package, and `apt policy` shows installed and available versions.

## Packages: getting information

If you want to know more about a package, you can query for information.

`apt show` displays metadata about a package, and `apt policy` shows installed and available versions.

`dpkg -L <package>` shows all files a certain installed package provides. Conversely, `dpkg -S <file>` shows what package a file comes from.

## Packages: getting information

If you want to know more about a package, you can query for information.

`apt show` displays metadata about a package, and `apt policy` shows installed and available versions.

`dpkg -L <package>` shows all files a certain installed package provides. Conversely, `dpkg -S <file>` shows what package a file comes from.

*Exercise:*

Find out what package `whoami` is from, what other programs are provided by this package and its dependencies.

## Automatic upgrades

In Debian, automatic upgrades are provided by `unattended-upgrades` package.

Its behavior defaults to automatically installing security updates.

It can be tuned by editing its configuration file:

```
owner@linux:~$ sudoedit /etc/apt/apt.conf.d/50unattended-upgrades
```

## Other sources for software

What if you're not satisfied with the official OS packages, what are other ways you can install software?

## Other sources for software

What if you're not satisfied with the official OS packages, what are other ways you can install software?

- Compatible pre-made packages (including closed source)
- Third-party repositories
- Building from source

## Compatible packages

`apt` is built upon `dpkg`, which expects **`.deb` (or "Debian")** packages.

If software provides `.deb` files, it's likely you can install them.

## Compatible packages

`apt` is built upon `dpkg`, which expects **`.deb` (or "Debian")** packages.

If software provides `.deb` files, it's likely you can install them.

- The architecture must match (i386 / amd64 / arm / ...)
- It may expect dependencies that are too old / too new / not available if it was built for another distribution or release

## Compatible packages

`apt` is built upon `dpkg`, which expects **`.deb` (or "Debian")** packages.

If software provides `.deb` files, it's likely you can install them.

- The architecture must match (i386 / amd64 / arm / ...)
- It may expect dependencies that are too old / too new / not available if it was built for another distribution or release

If you have a `.deb` file, you can install it with `apt`, pulling dependencies if needed (failing if impossible):

```
owner@linux:~$ apt install ./something.deb
```

## Third-party repositories

Installing a single pre-built package means upgrading it is fully manual. This is inconvenient and insecure.

## Third-party repositories

Installing a single pre-built package means upgrading it is fully manual. This is inconvenient and insecure.

Many projects provide a third-party repository. Adding it to apt's `sources.list` allows easy installation and automated updates.

In fact, many popular `.deb` installers automatically add their respective repository as well.

## Third-party repositories

Installing a single pre-built package means upgrading it is fully manual. This is inconvenient and insecure.

Many projects provide a third-party repository. Adding it to apt's `sources.list` allows easy installation and automated updates.

In fact, many popular `.deb` installers automatically add their respective repository as well.

This is a common way to install newer software versions in a static distribution.

## Aside: `/etc/**.d`

`/etc/` contains many subfolders of the form `something.d`.

This is a convention to allow splitting configuration files into multiple smaller files.

## Aside: `/etc/**.d`

`/etc/` contains many subfolders of the form `something.d`.

This is a convention to allow splitting configuration files into multiple smaller files.

`sources.list` follows this convention: besides the main file, everything matching `/etc/apt/sources.list.d/*.list` will be used.

It is a good practice to create new `.list` files for third-party repostitories.

## Adding a software source

Let's go to https://nginx.org/ and locate the repository for newest (mainline) version.

## Adding a software source

Let's go to https://nginx.org/ and locate the repository for newest (mainline) version.

Strings to add (for Ubuntu 18.04):

```
deb http://nginx.org/packages/mainline/ubuntu/ bionic nginx
deb-src http://nginx.org/packages/mainline/ubuntu/ bionic nginx
```

*Exercise:*

(Optional) If your OS is Debian but not Ubuntu 18.04, find the correct deb string.

Create a new file `/etc/apt/sources.list.d/nginx.list` with those lines.

## Package management: security matters

If we try to download package information from the newly-added repository, it will fail:

```
owner@linux:~$ sudo apt update
[...]
Err:5 http://nginx.org/packages/mainline/ubuntu bionic InRelease
  The following signatures couldn't be verified because the public key is not
available: NO_PUBKEY ABF5BD827BD9BF62
[...]
```

## Package management: security matters

If we try to download package information from the newly-added repository, it will fail:

```
owner@linux:~$ sudo apt update
[...]
Err:5 http://nginx.org/packages/mainline/ubuntu bionic InRelease
  The following signatures couldn't be verified because the public key is not
available: NO_PUBKEY ABF5BD827BD9BF62
[...]
```

All repository information must be signed by keys trusted by `apt`. We do not trust the key used by nginx.org yet.

To fix, we need to download it as instructed and add it to `apt`:

```
owner@linux:~$ wget https://nginx.org/keys/nginx_signing.key
[...]
owner@linux:~$ sudo apt-key add nginx_signing.key
OK
```

## Installing from a new repository

Before we can install software, we need to update our local `apt` database:

```
owner@linux:~$ sudo apt update
[...]
owner@linux:~$ sudo apt install -y nginx
[...]
```

We can confirm that `apt` installed a version from nginx.org by consulting `apt policy`:

```
owner@linux:~$ nginx -v
nginx version: nginx/1.15.8
owner@linux:~$ apt policy nginx
[...]
 *** 1.15.8-1~bionic 500
    500 http://nginx.org/packages/mainline/ubuntu bionic/nginx amd64 Packages
[...]
```

## Ubuntu-specific: PPAs

Personal Package Archives are a special class of third-party repositories indigenous to Ubuntu, hosted by Canonical.

They are easier to set up than manually adding the `deb` lines to `sources.list` and manually importing keys.

```
owner@linux:~$ sudo add-apt-repository ppa:atareao/telegram
```

Otherwise, they function identical to 3rd-party repositories.

## Installing from source

Some software we want may not have a ready binary package.

In that case, we must download its source, build and install it.

## Installing from source

Some software we want may not have a ready binary package.

In that case, we must download its source, build and install it.

How to do that differs greatly, but common workflow is:

- Download the source
- Check INSTALL or README files for requirements
- Install dependencies
- Run a configure script
- Run `make` to build the software
- Run `sudo make install` to install it

## Guessing dependencies

If you're lucky, dependencies are specified in terms of package names.

## Guessing dependencies

If you're lucky, dependencies are specified in terms of package names.

Often, they are not; as an example, what if the configure script complains about a missing library `foo`? Or the build step can't find `foo.h`?

## Guessing dependencies

If you're lucky, dependencies are specified in terms of package names.

Often, they are not; as an example, what if the configure script complains about a missing library `foo`? Or the build step can't find `foo.h`?

1. Remember that `apt search` is your friend.
2. Debian packages for libraries are normally called `libfoo*`.
3. For compilation, you need header files for libraries. By convention, those are named `libfoo-dev`.

## Guessing dependencies

If you're lucky, dependencies are specified in terms of package names.

Often, they are not; as an example, what if the configure script complains about a missing library `foo`? Or the build step can't find `foo.h`?

1. Remember that `apt search` is your friend.
2. Debian packages for libraries are normally called `libfoo*`.
3. For compilation, you need header files for libraries. By convention, those are named `libfoo-dev`.

When everything else fails, Google it!

## Example: install Singularity

We'll try installing Singularity

*Exercise:*

Follow instructions to install Singularity at https://www.sylabs.io/guides/2.6/user-guide/installation.html#install-a-specific-release

Use `VER="2.6.1"`

Dependencies:

```
build-essential squashfs-tools libarchive-dev
```

## Example: install Singularity

We'll try installing Singularity

*Exercise:*

Follow instructions to install Singularity at https://www.sylabs.io/guides/2.6/user-guide/installation.html#install-a-specific-release

Use `VER="2.6.1"`

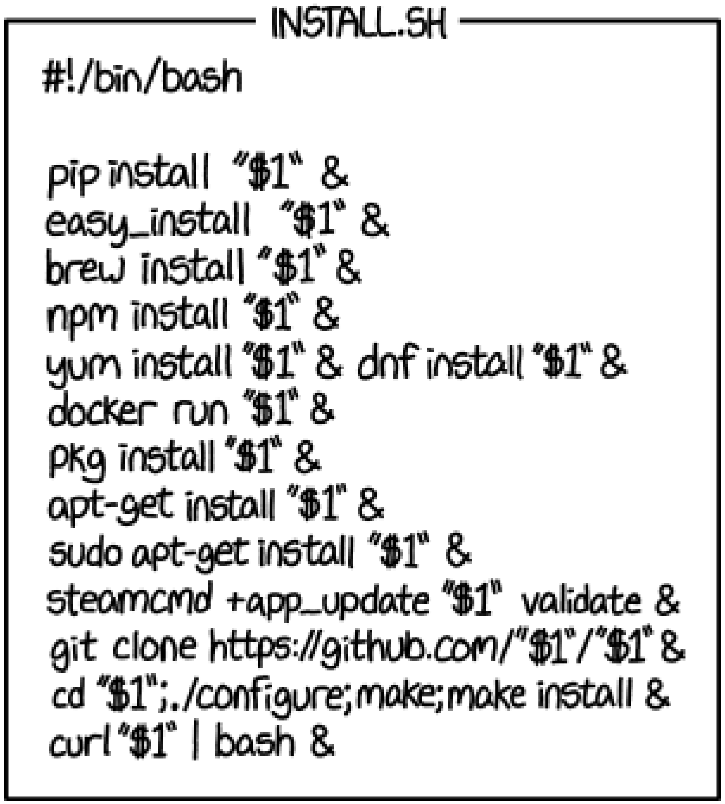
Dependencies:

```
build-essential squashfs-tools libarchive-dev
```

Let's test it!

```
owner@linux:~$ singularity run docker://godlovedc/lolcow
```

## Bonus: Universal Install Script


INSTALL.SH

```
#!/bin/bash

pip install "$1" &
easy_install "$1" &
brew install "$1" &
npm install "$1" &
yum install "$1" & dnf install "$1" &
docker run "$1" &
pkg install "$1" &
apt-get install "$1" &
sudo apt-get install "$1" &
steamcmd +app_update "$1" validate &
git clone https://github.com/"$1"/"$1" &
cd "$1";./configure;make;make install &
curl "$1" | bash &
```

*Image credit:* XKCD #1654

## User management

A **user** is a basic unit of access control; it has a set of credentials to access the system and **owns** some files on it.

A **group** is a collection of users to facilitate shared access to resources. A user can belong to many groups but one group is considered primary.

As an admin, you may need to add/delete users and change their rights and passwords.

This intro only covers the basic scenario of local users and groups.

## User and group information

For local users and groups, their information is stored in:

- `/etc/passwd` - users, their primary groups and basic settings
- `/etc/shadow` - hashed user passwords and expiration data
- `/etc/group` - group and their members

*Advice:*

Do not edit these files directly; use specialized utilities.

# User and group information

For local users and groups, their information is stored in:

- `/etc/passwd` - users, their primary groups and basic settings
- `/etc/shadow` - hashed user passwords and expiration data
- `/etc/group` - group and their members

*Advice:*

Do not edit these files directly; use specialized utilities.

You can use **`getent <database>`** to access these lists, or `getent <database> <id>` to get a specific record.

```
owner@linux:~$ getent passwd owner
owner:x:1001:1001:,,,:/home/owner:/bin/bash
```

## Changing passwords

Any user can change their own password with `passwd`, if they can provide their current one:

```
owner@linux:~$ passwd
Changing password for owner.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

## Changing passwords

Any user can change their own password with `passwd`, if they can provide their current one:

```
owner@linux:~$ passwd
Changing password for owner.
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Superuser can change any account's password without knowing the current one:

```
owner@linux:~$ sudo passwd owner
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

## Aside: home folder encryption

If a user's home folder is encrypted, the decryption key is protected with their password.

In case the user changes their own password, the decryption key is automatically re-encrypted using the provided old password.

In case of an admin password change, the user's old password will still be needed to unlock the decryption key.

## Editing users, groups and their relation

To add a user, use `adduser <user>`; it will interactively ask for information.

By default, it will automatically create a new home folder and a primary group with the same name and one member.

## Editing users, groups and their relation

To add a user, use `adduser <user>`; it will interactively ask for information.

By default, it will automatically create a new home folder and a primary group with the same name and one member.

To add a group, use `addgroup <group>`.

To add an existing user to a group, `adduser <user> <group>`

## Editing users, groups and their relation

To add a user, use **`adduser <user>`**; it will interactively ask for information.

By default, it will automatically create a new home folder and a primary group with the same name and one member.

To add a group, use **`addgroup <group>`**.

To add an existing user to a group, **`adduser <user> <group>`**

To remove a user from a group, use **`deluser <user> <group>`**

Users and groups are deleted with **`deluser <user>`** and **`delgroup <group>`**

*Reference:* `man adduser`, `man deluser`

## The `skel`-eton of a home

When a new user's home folder is created, it's possible to automatically populate its contents.

Contents of `/etc/skel` are used as a template.

They are copied to the new home folder and changed to be owned by the new user.

## Exercise: adding a user

*Exercise:*

1. Create a user `test_user`
2. Change to that user with `su - test_user`
3. Try using `sudo`
4. Add the user to `sudo` group
5. Try again
6. Delete the user (with `--remove-home`)

*Note:*

A change in user's groups only fully applies if they log out

## Examining your system

An important part of Linux administration is being able to gather information about the running system.

In this section multiple ways to do that are presented.

## What's the hardware?

Common commands to gather info about the hardware:

- `cat /proc/cpuinfo` and `lscpu` present CPU information
- `lspci` shows devices connected to the system bus
- `lsusb` shows devices connected via USB
- `lsblk` shows storage devices and their partitions
- `lshw` shows a hierarchical view of all hardware
- `dmidecode` shows information reported by the motherboard

*Exercise:*

Try them all out; they should be run as `root`

## What are the network settings?

`ip` is the command to access networking information in modern Linux systems.

For example, `ip a` shows addresses assigned to all interfaces.

Basic network settings are accessible through `/etc/networks`.

## Are we running out of space?

`df -h` (for "disk free") shows how much space is used per filesystem.

*Note:*

Ext filesystems reserve part of available space for emergency use by `root`. It will show up as used.

The amount can be changed on the fly with `tune2fs`.

## Are we running out of space?

`df -h` (for "disk free") shows how much space is used per filesystem.

*Note:*

Ext filesystems reserve part of available space for emergency use by `root`. It will show up as used.

The amount can be changed on the fly with `tune2fs`.

`free -h` shows information about used memory.

Note the difference between "free" (unused) and "available" (ready to be emptied) memory.

As such, low *free* memory is not cause for concern.

*Reference:* Reserved space for root on a filesystem - why?

## What's eating all the space?

The command `du` (for "disk usage") allows for calculation of real size of folders.

- `du -h <path>` calculates all folder sizes recursively from provided path.
- `-s` provides a summary: only the total for the path.
- `-d <level>` provides numbers up to `<level>` deep
- `-x` stops `du` from crossing into different filesystems (very useful for `/`)

*Exercise:*

Try `du -h -d 1 ~` to measure use of your home folder

## What's eating all the space?

The command `du` (for "disk usage") allows for calculation of real size of folders.

- `du -h <path>` calculates all folder sizes recursively from provided path.
- `-s` provides a summary: only the total for the path.
- `-d <level>` provides numbers up to `<level>` deep
- `-x` stops `du` from crossing into different filesystems (very useful for `/`)

*Exercise:*

Try `du -h -d 1 ~` to measure use of your home folder

A friendlier interface for the same task is `ncdu`.

## How's the load?

`uptime` shows how long the system has been running, as well as how busy the CPU is.

```
owner@linux:~$ uptime
 10:04:17 up  1:03,  1 user,  load average: 0.08, 0.02, 0.01
```

## How's the load?

`uptime` shows how long the system has been running, as well as how busy the CPU is.

```
owner@linux:~$ uptime
 10:04:17 up  1:03,  1 user,  load average: 0.08, 0.02, 0.01
```

In this example, the system has been running for 1h3m.

## How's the load?

`uptime` shows how long the system has been running, as well as how busy the CPU is.

```
owner@linux:~$ uptime
 10:04:17 up  1:03,  1 user,  load average: 0.08, 0.02, 0.01
```

In this example, the system has been running for 1h3m.

Load average can be seen as a ratio of scheduled CPU tasks vs idle time in a given time window (1, 5, and 15 minutes).

Ideally, this ratio should not be much more than amount of virtual cores in the system.

## How's the load?

`uptime` shows how long the system has been running, as well as how busy the CPU is.

```
owner@linux:~$ uptime
 10:04:17 up  1:03,  1 user,  load average: 0.08, 0.02, 0.01
```

In this example, the system has been running for 1h3m.

Load average can be seen as a ratio of scheduled CPU tasks vs idle time in a given time window (1, 5, and 15 minutes).

Ideally, this ratio should not be much more than amount of virtual cores in the system.

It also lists number of logged-in users. They are listed by `who`.

## What's running?

`ps` is a utility to list running processes in the system.

Its output is highly tunable; my personal preference is `ps aux` for all processes on the system.

## What's running?

`ps` is a utility to list running processes in the system.

Its output is highly tunable; my personal preference is `ps aux` for all processes on the system.

A more intuitive overview can be obtained with `pstree -u`, which shows processes by parent-child relation.

## What's running?

`ps` is a utility to list running processes in the system.

Its output is highly tunable; my personal preference is `ps aux` for all processes on the system.

A more intuitive overview can be obtained with `pstree -u`, which shows processes by parent-child relation.

For automation purposes, `pgrep` is also useful.

## What's hogging all the CPU / RAM?

`top` is a monitor of system resources.

It allows to quickly see what's using up system resources.

A nicer modern version is `htop`.

## What is keeping things open?

**`lsof`** is a tool to see all open files and file-like objects by processes.

**`lsof <path>`** shows processes keeping files under `<path>` open.

**`lsof -i`** displays all open network ports.

## How to see logs?

As mentioned before, most logs are stored under **`/var/log/`**

Logs are periodically "rotated", with older ones having `.1`, `.2`, … appended to them. Oldest ones may be compressed.

- To see plain-text logs, `less` is recommended.
- **`zless`** allows viewing compressed archives (`.N.gz`) directly.
- **`tail -f`** allows monitoring new log entries as they appear.
- **`dmesg`** shows kernel logs since boot.

However, on modern systems using `systemd` many logs are collected (or at least mirrored) in a single database accessed through **`journalctl`**.

## Working with services

Unlike programs we run directly, **services** (or **daemons**) are run in the background, to be ready to respond to tasks.

A web server, a database server, or SSH server are examples of services.

## Working with services

Unlike programs we run directly, **services** (or **daemons**) are run in the background, to be ready to respond to tasks.

A web server, a database server, or SSH server are examples of services.

To understand services, we first need to undestand `init`.

When Linux boots, the kernel starts process ID 1, `init`, which is responsible for starting and managing all other processes.

*Advice:*

It's very helpful to see `pstree` output here.

Init has lists of various services and rules as to when and how to run them.

## Init and `systemd`

Init systems of old were collection of scripts that would start and stop services and were controlled with `service` command.

## Init and `systemd`

Init systems of old were collection of scripts that would start and stop services and were controlled with `service` command.

In recent years, the majority of Linux distributions switched to using `systemd`, which is, among other things, an Init system.

It has controversial reputation, but is the current reality.

## Init and `systemd`

Init systems of old were collection of scripts that would start and stop services and were controlled with `service` command.

In recent years, the majority of Linux distributions switched to using `systemd`, which is, among other things, an Init system.

It has controversial reputation, but is the current reality.

Under `systemd`, services are called **units** and are controlled with the **`systemctl`** command.

## Units

Units in `systemd` are stored in various locations; typically, OS-provided units are stored in `/lib/systemd/system`, and **`/etc/systemd/system`** is a good location for custom units.

Unit (service) files are small configuration files describing how and when a service is run.

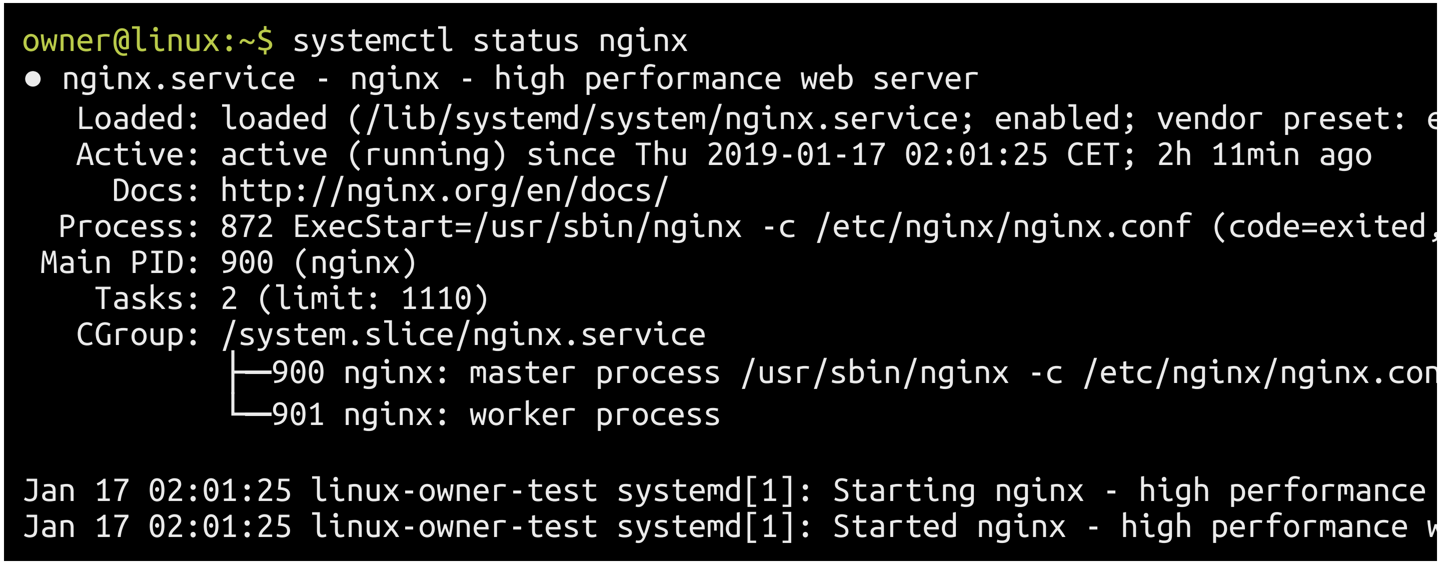
The list of units can be obtained with `systemctl list-unit-files`.

Services can be **enabled or disabled**, which reflects whether they will be automatically started (e.g. on boot).

Services can be in started (running) or stopped state.

## Checking services

To check a service's state, use `systemctl status <service>`. It prints its current state plus a small excerpt of its log.

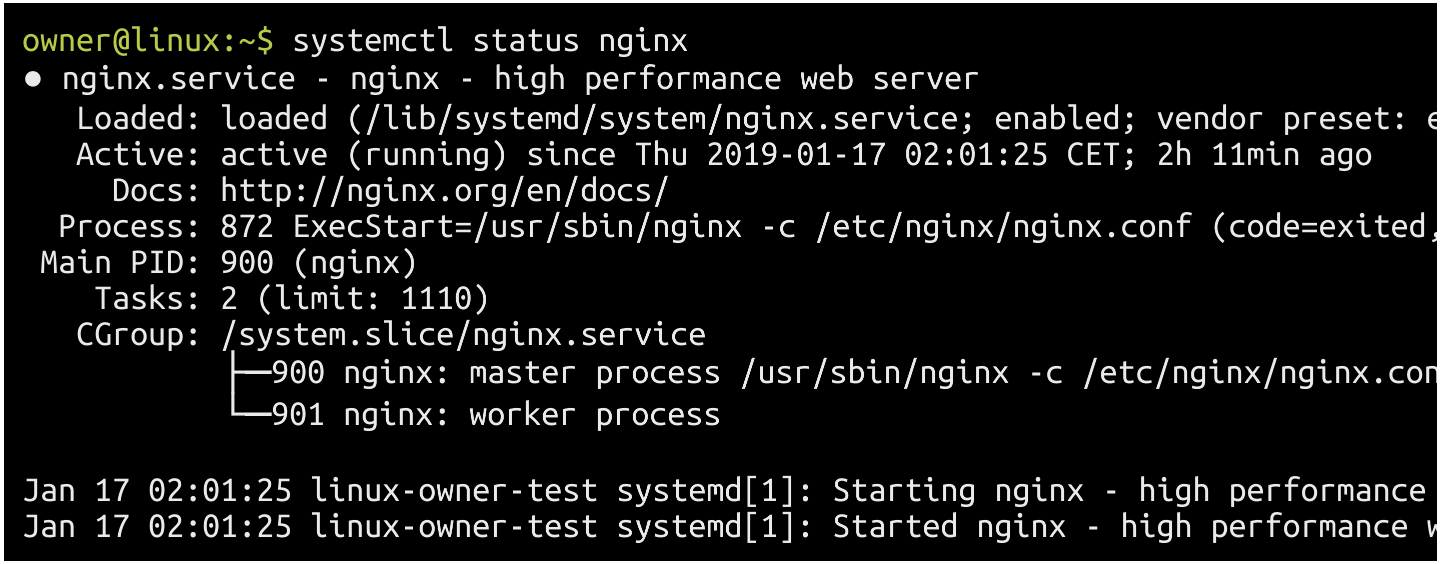


```
owner@linux:~$ systemctl status nginx
● nginx.service - nginx - high performance web server
   Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: e
   Active: active (running) since Thu 2019-01-17 02:01:25 CET; 2h 11min ago
     Docs: http://nginx.org/en/docs/
  Process: 872 ExecStart=/usr/sbin/nginx -c /etc/nginx/nginx.conf (code=exited,
 Main PID: 900 (nginx)
    Tasks: 2 (limit: 1110)
   CGroup: /system.slice/nginx.service
           ├─900 nginx: master process /usr/sbin/nginx -c /etc/nginx/nginx.con
           └─901 nginx: worker process

Jan 17 02:01:25 linux-owner-test systemd[1]: Starting nginx - high performance 
Jan 17 02:01:25 linux-owner-test systemd[1]: Started nginx - high performance w
```

## Checking services

To check a service's state, use `systemctl status <service>`. It prints its current state plus a small excerpt of its log.



```
owner@linux:~$ systemctl status nginx
● nginx.service - nginx - high performance web server
   Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: e
   Active: active (running) since Thu 2019-01-17 02:01:25 CET; 2h 11min ago
     Docs: http://nginx.org/en/docs/
  Process: 872 ExecStart=/usr/sbin/nginx -c /etc/nginx/nginx.conf (code=exited,
 Main PID: 900 (nginx)
    Tasks: 2 (limit: 1110)
   CGroup: /system.slice/nginx.service
           ├─900 nginx: master process /usr/sbin/nginx -c /etc/nginx/nginx.con
           └─901 nginx: worker process

Jan 17 02:01:25 linux-owner-test systemd[1]: Starting nginx - high performance
Jan 17 02:01:25 linux-owner-test systemd[1]: Started nginx - high performance w
```

Full log of a service can be seen with `journalctl -u <service>`

## Controlling services

To change service state (requires `root`):

- `systemctl start <service>` attempts to start a service
- `systemctl stop <service>` stops a service
- `systemctl restart <service>` stops and starts a service
- `systemctl reload <service>` allows changing service configuration without restarting it. Not all units support it.
- `systemctl enable <service>` ensures a service will be started automatically (e.g. on reboot)
- `systemctl disable <service>` prevents a service from starting automatically

After (re)starting or reloading a service it's highly recommended to check its status, in case some fatal error occurred at start.

## Service configuration

For most services, the configuration is somewhere in `/etc`

For example, `sshd` configuration is at `/etc/ssh/sshd.conf`

It's important to remember that most services require a reload or a restart after changing their configuration.

## Simplest service unit

Let's create a simple service that runs a shell script.

*Exercise:*

As root, add a file `/opt/beep.sh`:

```
#!/bin/bash
start() {
  echo 'Beep!'
}

stop() {
  echo 'Boop!'
}

case $1 in
  start|stop) "$1" ;;
esac
```

Make it executable.

## Simplest service unit

*Exercise:*

- Add a file `/etc/systemd/system/beep.service`:

```
[Unit]
Description=Beep-boop

[Service]
Type=oneshot
ExecStart=/opt/beep.sh start
ExecStop=/opt/beep.sh stop
RemainAfterExit=yes

[Install]
WantedBy=multi-user.target
```

- Tell `systemd` about the unit with `systemctl daemon-reload`
- Try controlling it

*Reference:* How to write a startup script for systemd

## Scheduled and regular jobs

If you need some script to run at a later date, or at regular intervals:

- `at` provides simple scheduling for later execution
- `cron` allows setting up repeating tasks

Working with `cron` requires understanding its (not very obvious) configuration format.

*Reference:* A Beginners Guide To Cron Jobs

## Security considerations

OS security is a large topic worthy of a separate course.

We will only briefly touch some basic topics.

## Importance of security updates

Security updates exist to eliminate found vulnerabilities in software.

Vulnerabilities differ wildly in potential for exploitation, but it's still sensible to close holes as soon as possible.

Updates that require a reboot should not be put off indefinitely either.

## Login security

Use good passwords. `apg` is a generator of strong passwords that are easy to remember.

Safeguard your SSH keys. Do not use non-encrypted keys or agent forwarding unless strictly necessary.

## Login security

Use good passwords. `apg` is a generator of strong passwords that are easy to remember.

Safeguard your SSH keys. Do not use non-encrypted keys or agent forwarding unless strictly necessary.

If possible, require public key authentication for SSH.

Use mechanisms like `fail2ban` to mitigate brute-force attempts.

## Sensitive files and access rights

Denying others read permission on sensitive files (private keys, security configuration, etc.) is a key protection.

Do not try to fix access problems with a permissive `chmod`. This can do more harm than good.

## Encryption

Full-disk encryption protects against physical-access attacks.

Home folder encryption protects users against other users (including admins).

Both come at the expense of some performance and risk of data loss if encryption password is lost.

## Encryption IRL

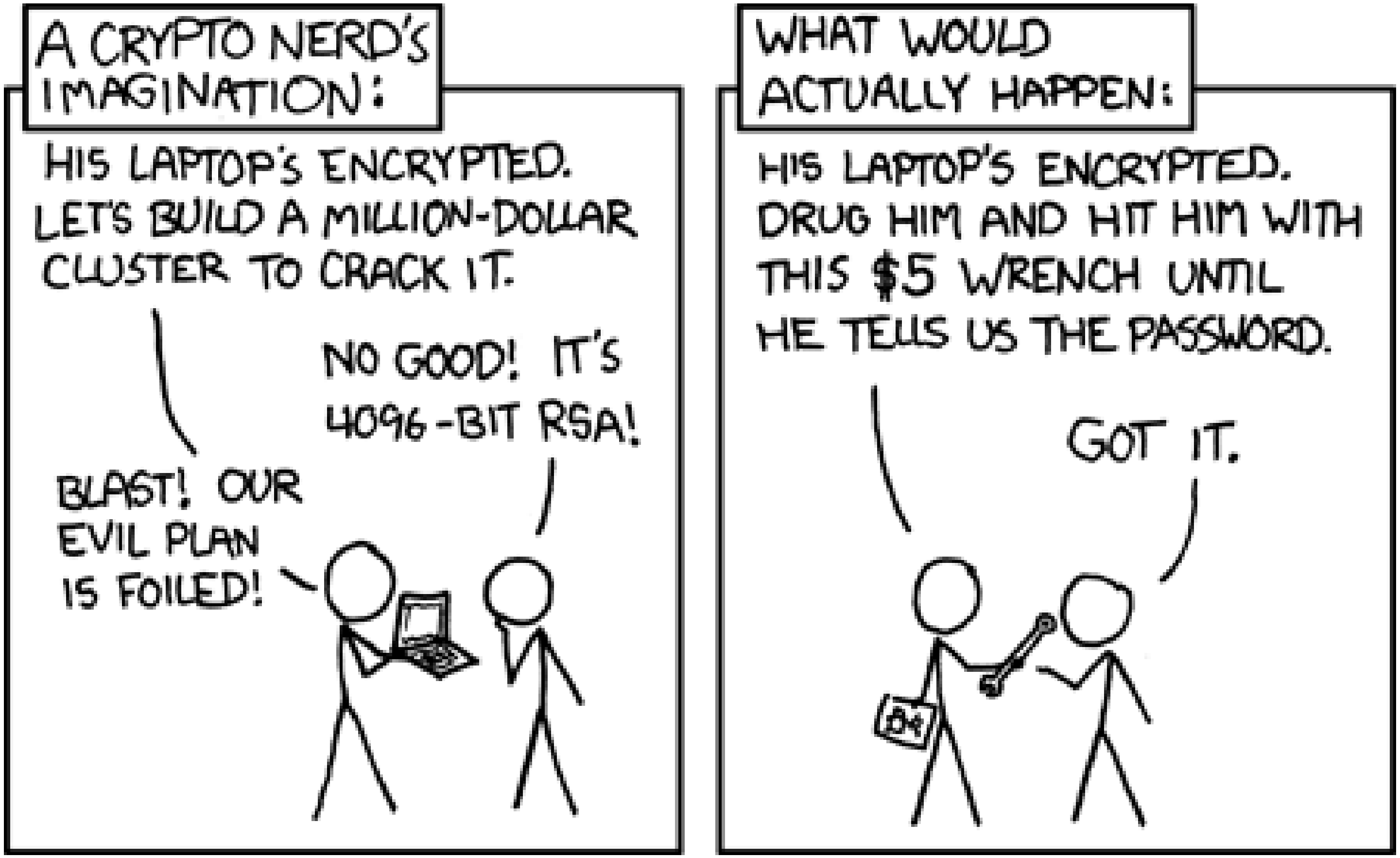

A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.

*Image credit:* [XKCD #538]

## Firewall

A firewall helps prevent unauthorized programs from communicating.

You will have to designate services that are allowed to communicate yourself, but it will prevent any surprises.

Ubuntu uses `ufw` as the firewall frontend, but low-level back-end is `iptables`.

Packages have a way to add exceptions to firewall, so for standard ports you usually don't need to manually intervene.

In desktop Ubuntu the firewall is disabled by default.

## SELinux / AppArmor

Sometimes, programs have to run as root to achieve one specific thing, and can be a risk if they can be exploited.

There are kernel modules like SELinux that restrict activity of processes based on configurable rules, even for `root`.

This allows to follow least-privilege principles at the expense of complex-to-maintain configuration.

## Basic Troubleshooting

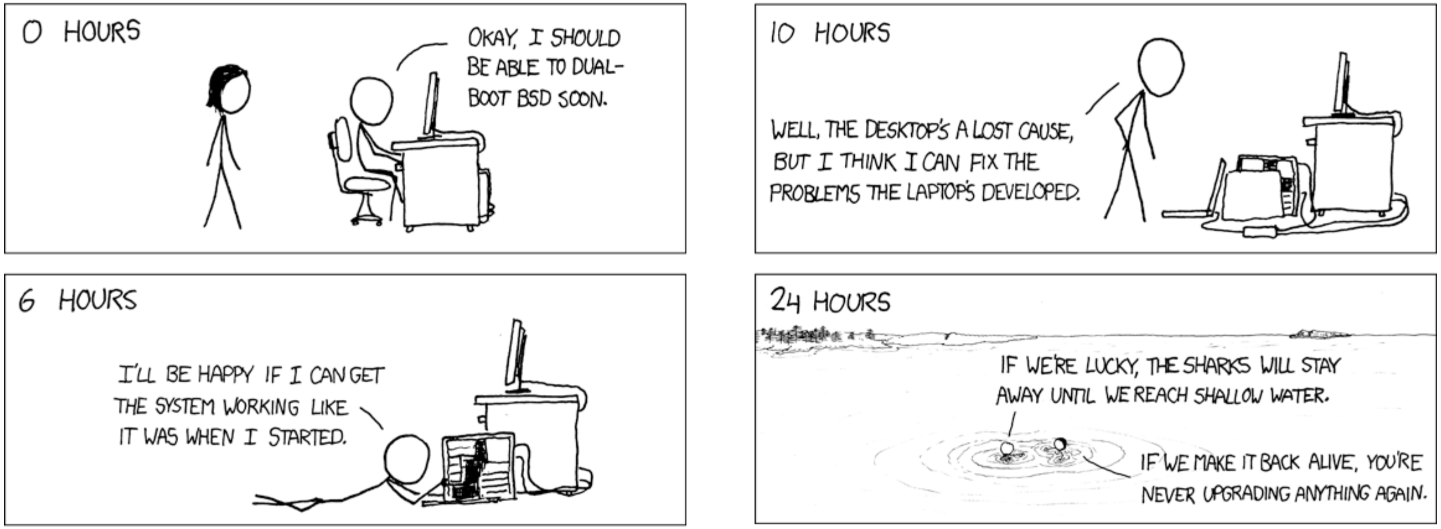

0 HOURS
OKAY, I SHOULD BE ABLE TO DUAL-BOOT BSD SOON.
10 HOURS
WELL, THE DESKTOP'S A LOST CAUSE, BUT I THINK I CAN FIX THE PROBLEMS THE LAPTOP'S DEVELOPED.
6 HOURS
I'LL BE HAPPY IF I CAN GET THE SYSTEM WORKING LIKE IT WAS WHEN I STARTED.
24 HOURS
IF WE'RE LUCKY, THE SHARKS WILL STAY AWAY UNTIL WE REACH SHALLOW WATER.
IF WE MAKE IT BACK ALIVE, YOU'RE NEVER UPGRADING ANYTHING AGAIN.

*Image credit:* XKCD #349

## Verbosity

When some program fails, it helps to ask it to produce more low-level information, either in the output or in the logs.

Many programs have switches or configuration options for that, often `-v` (for **verbose**).

There can even be multiple verbosity levels.

## Spying on programs

If a program is running into an unhelpful error, you can try watching what it's doing under the hood.

`strace` is a general-purpose debugger (requiring root) that monitors all system calls made by a process.

In practice, that means being able to trace e.g. all file access.

## Dealing with broken GUI

If your graphical environment can't function normally (hangs, or you can't log in), you can try accessing text-mode consoles.

This is done with Ctrl+Alt+F1 through Ctrl+Alt+F12 to switch between them.

*Exercise:*

Try accessing your TTYs (won't work over SSH)

## Dealing with broken boot

If your system can't boot normally, you can try booting it in recovery mode.

Hold Shift or mash Esc during boot (after BIOS) to bring up the GRUB menu.

From there, find and select recovery mode; the system will boot in reduced "single user" mode, and you can select to start a `root` shell to try and fix the system or apply other recovery actions.

You need to be physically present to attempt this. This can be used to circumvent / recover passwords as well.

## Dealing with *really* broken boot

If you can't boot even in recovery, you can boot from an installation USB with a live Linux system, mount the filesystem of the installed Linux and make changes that way.

Again, this requires physical presence, but fully bypasses any and all security of the system (except encryption).

## Where to get help?

If you know the command, or the config file that you're trying to use, `man` is your friend.

## Where to get help?

If you know the command, or the config file that you're trying to use, `man` is your friend.

Also, Google is your friend! There's no shame in learning what you don't know yet or looking up what you don't remember.

## Where to get help?

If you know the command, or the config file that you're trying to use, `man` is your friend.

Also, Google is your friend! There's no shame in learning what you don't know yet or looking up what you don't remember.

Some notable sites that are very helpful:

- StackExchange network: https://stackoverflow.com/, https://superuser.com/, https://askubuntu.com/, https://unix.stackexchange.com/, https://serverfault.com/
- DigitalOcean tutorials: https://www.digitalocean.com/community/tutorials/
- nixCraft tutorials: https://www.cyberciti.biz/

## Where to get help?

If you know the command, or the config file that you're trying to use, `man` is your friend.

Also, Google is your friend! There's no shame in learning what you don't know yet or looking up what you don't remember.

Some notable sites that are very helpful:

- StackExchange network: https://stackoverflow.com/, https://superuser.com/, https://askubuntu.com/, https://unix.stackexchange.com/, https://serverfault.com/
- DigitalOcean tutorials: https://www.digitalocean.com/community/tutorials/
- nixCraft tutorials: https://www.cyberciti.biz/

When searching for solutions, specifying your OS version is helpful. Old advice sometimes does not apply to newer systems!

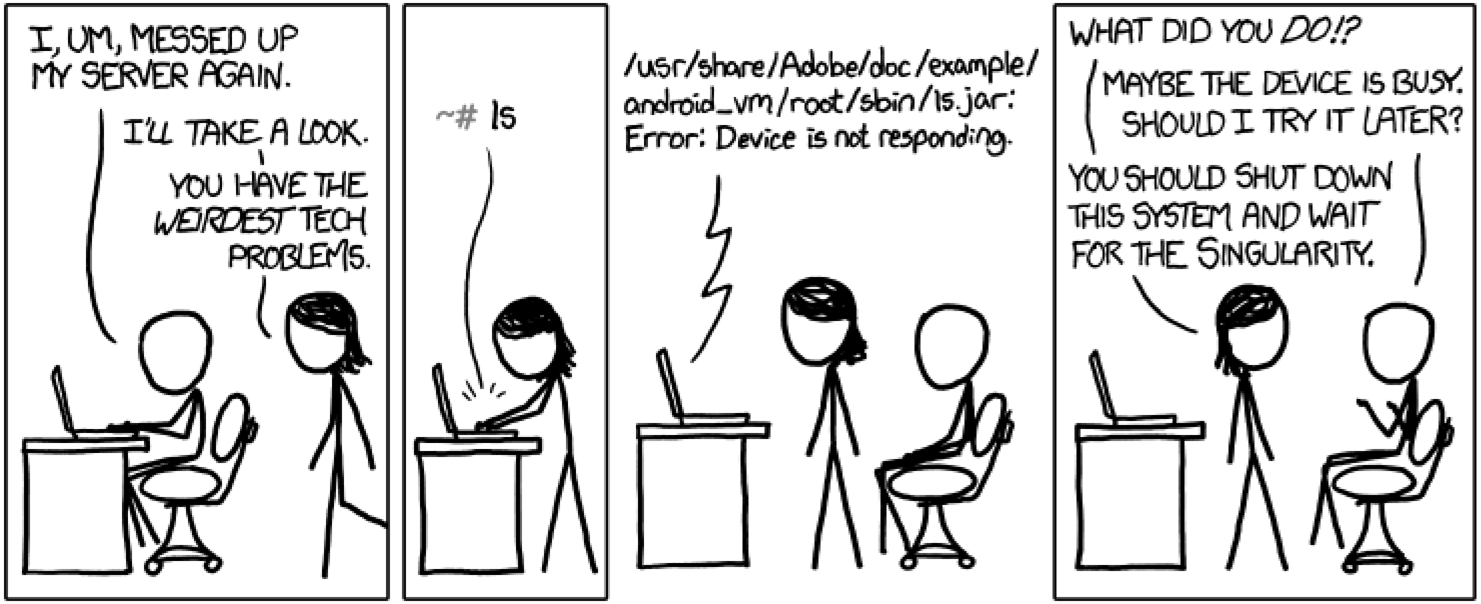
## That's it!

Go forth and administrate!

## That's it!

### Go forth and administrate!

..try not to break stuff, though.


I, UM, MESSED UP MY SERVER AGAIN.
I'LL TAKE A LOOK.
YOU HAVE THE WEIRDEST TECH PROBLEMS.
~# ls
/usr/share/Adobe/doc/example/android_vm/root/sbin/ls.jar: Error: Device is not responding.
WHAT DID YOU DO!?
MAYBE THE DEVICE IS BUSY. SHOULD I TRY IT LATER?
YOU SHOULD SHUT DOWN THIS SYSTEM AND WAIT FOR THE SINGULARITY.
Image credit: XKCD #1084