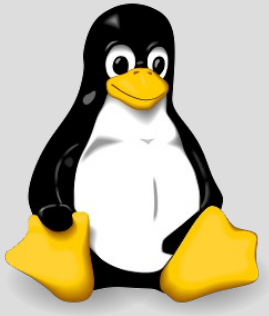
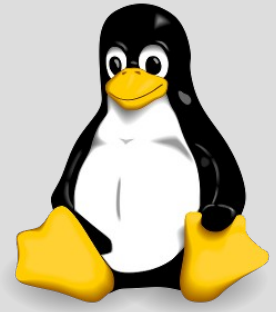
- Log into the Moodle site
- Enter the “Week 3” area (button 3)
- At 14:00, choose “Daily Quiz 2”
- Answer the multiple choice quiz (you have 10 minutes)



Exploring the system, investigating hardware & system resources

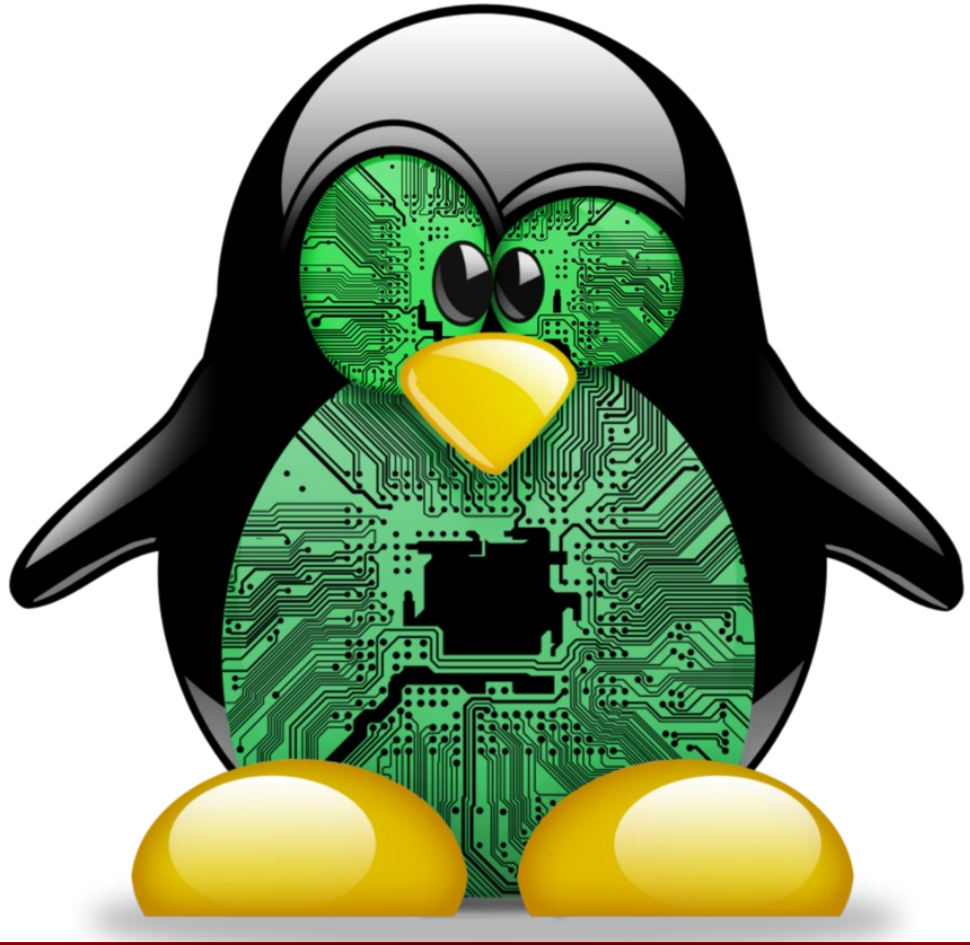


J. M. P. Alves

Laboratory of Genomics & Bioinformatics in Parasitology

Department of Parasitology, ICB, USP

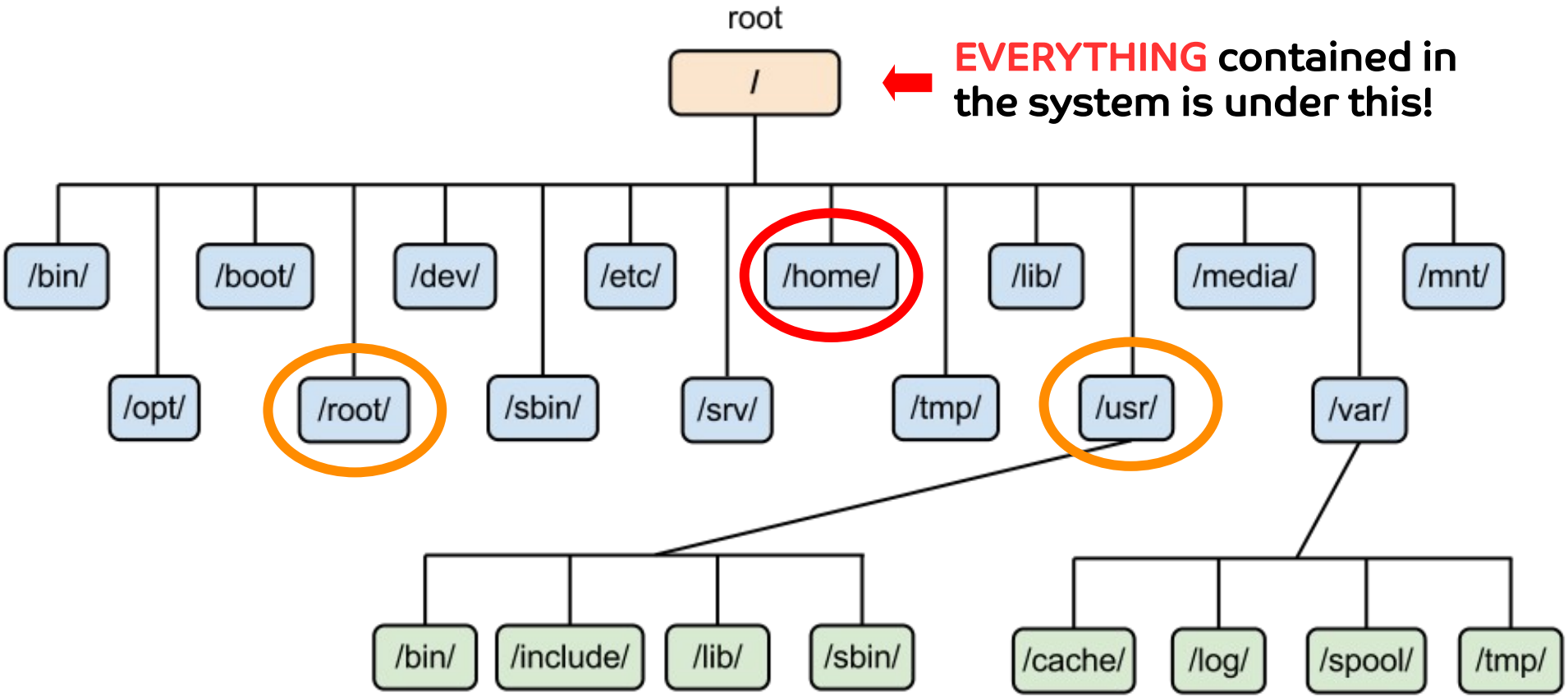
Organization of a Linux system



Organization of a Linux system

- In Unix, **everything** is either a **file** or a **process** (running program)
  - Files
  - Screen
  - Keyboard
  - Disks
  - Printer
  - Speakers
  - CPU (main processor, chip)
  - Etc. Etc.
- Files are organized in a hierarchical structure shaped like a **tree**
- A directory can have files and/or other directories inside itself
- The top level of the hierarchy is the **root directory**, represented by **/**

File system structure



File system structure

- **/bin** – common programs, shared by the system, the system administrator and the users
- **/boot** – the startup files and the kernel. In some recent distributions also grub data. Grub is the GRand Unified Boot loader
- **/dev** – contains references to all the CPU and peripheral hardware, which are represented as files with special properties
- **/etc** – most important system configuration files are in /etc
- **/home** – this is where user data lives
- **/initrd** – on some distributions, information for booting

File system structure

- `/root` – the administrative user's home directory. Mind the difference between `/`, the root directory and `/root`, the home directory of the root user
- `/sbin` – programs for use by the system and the system administrator
- `/tmp` – temporary space for use by the system, cleaned upon reboot
- `/usr` – programs, libraries, documentation etc. for all user-related programs
- `/var` – storage for all variable files and temporary files created by users, such as log files, the mail queue, the print spooler area, space for temporary storage of files downloaded from the Internet, or to keep an image of a CD before burning it

File system structure

- **`/lib`** – library files, includes files for all kinds of programs needed by the system and the users
- **`/lost+found`** – every partition has a lost+found in its upper directory; files that were saved during failures are here
- **`/misc`** – for miscellaneous purposes
- **`/mnt`** – standard mount point for external file systems, e.g. a CD-ROM or a digital camera
- **`/net`** – standard mount point for entire remote file systems
- **`/opt`** – typically contains extra and third party software
- **`/proc`** – a virtual file system containing information about system resources

Two very important “files”

. and . .

- These are actually **not real files**, but abstractions that represent a **relative position in relation to the current directory** where the user is
- The first one (.) represents the **current directory itself**
- The second (. .) represents **the directory immediately above**, in the tree, to the one where the user is
- The . . **can be “concatenated”** by using a slash: e.g. . . / . . / (two levels above), . . / . . / . . / (three levels above), and so on –as you must have noticed, the / is the directory separator in Unix (in Windows, it is \)

The period (.) again

- Files whose names start with a period (`.`) are **hidden** from the **ls** command (and from a graphical file manager too!) unless the user explicitly asks for them
- `.bashrc`, for example, is a file that is not seen when one runs `ls` in the home directory of a user
- To see **all** files, including the hidden ones, use the `-a` option with `ls`
- **Try it!** Go to your home directory and run `ls` without any options; look at what you got. Then run `ls -a` and compare with the previous output

Absolute x relative path

Path?

- In computing, the **path** is the **address** in the system for a file or directory (i.e., the path you have to follow to get to the file)
- An **absolute path** (also called a **full path**) is an address **relative to the root directory** (i.e., the directory at the very top of the file system and which contains all other directories and files)
- A **relative path** is an address **relative to the current directory** (i.e., the directory in which the user is currently working)

Absolute x relative path

How to tell?

VERY easy! An **absolute path always** starts with the **/** character (but beware of the **tilde expansion...** `~/data` is the same as `/home/user/data`)

Examples:

Absolute

```
/home/aluno/Documents
/
/bin/ls
~/Downloads
```

Relative

```
aluno/Documents
../data/test
./dir1/things/
./
```

Quiz time!

Go to the course page and choose **Quiz 2**

First: Where are you?

pwd

- Short for "**print working directory**"
- Tells you which directory you are currently working in
- Very useful when you are inside many sub-sub-sub-directories and got lost... or when you need to copy the full absolute path of something in the file system

Moving around

cd

- To change the directory where you are, use the **cd** command (short for **"change directory"**)
- For example: **cd /etc/apache2** (full path!) moves the user from wherever they are to directory `/etc/apache2`
- Or: **cd a_sub_dir** (relative path!) moves the user to a subdirectory (called `a_sub_dir`) of their current one
- By itself, **cd** goes to the user's home (e.g., **/home/mary**)
- **cd -** (minus sign) goes back to the directory where the user was before being in the current one

Remember!

- Use the **TAB** key for auto-completion of what you are typing!
- Use TAB **twice quickly** to see alternatives, if the first TAB got stuck on an ambiguity!
- Use the **arrow keys** to navigate the command history and reuse what you typed earlier!

Let's try it!

- Log into the remote server, if are not there already
- Run `pwd`
- What do you see? Your home directory's absolute path!
- Now run `cd /home/jmalves`
- You are now in MY user directory (behave yourself!)
- Run `cd` (by itself, this time), and then `pwd`
- What happened? Where did you go?
- Now run `cd -` (again, that is the minus sign)
- Where did you go now?

Let's practice some more...

Being in the **remote server**, try the following commands:

`ls /usr/bin/n*e` ← `*` = zero or more of any character

`ls /usr/bin/l[a-f]*` ← `[]` = delimits a list of single characters (with `-` to list all possibilities from the first to last)

`ls /usr/bin/l[a-f]?` ← `?` = exactly any one character

`ls /usr/bin/l[acb]*` ← will list only files with names starting with la, lc, and lb (followed by zero or more other characters)

`ls /usr/bin/w{3m,a,.}*` ← `{}` = a comma-separated list of patterns to match

Let's practice some more...

```
ls /usr/bin/xz{c{a..n},g}*
```

`{}` can be nested; now the sequence specifier is `..` and not `-` anymore!

```
ls /usr/bin/xz{c{n..a},g}*
```

`{}` the sequence can go either way: up or down

```
ls /usr/bin/l[f-a]*
```


will not work, must be `[a-f]`

```
echo {3..-2}
```


will print `3 2 1 0 -1 -2`

```
echo {-2..3}
```

will print `-2 -1 0 1 2 3`

```
echo {0..20..2}
```

`{x..y..z}` = will return from x to y, with increments (or decrements) of z

Quiz time!

Go to the course page and choose **Quiz 3**

Type the following three commands:

```
wget http://lgbp.online/PE7.zip
unzip PE7.zip
cd PE7
```

Now you do it!

Go to the course site and enter **Practical Exercise 7**

Follow the instructions to answer the questions in the exercise

Remember: in a PE, you should do things in practice before answering the question!

Ctrl+c/Ctrl+v in the shell?

- Paste, in Gnome Terminal, is **`Shift+Insert`** (that is the key called **`Insert`** or **`Ins`** in the keyboard)

- To copy something, select it with the mouse, as you would in a graphical application (a **double click** selects a **whole word**; a **triple click**, a **whole line**)

- Now, the trick: after selecting, you don't need to do anything! Linux has a **special buffer memory area** that already copies anything that is selected (that works anywhere, not just the shell)

- To paste what you copied, you can also press the mouse's **middle button** (if yours doesn't have one, pressing both buttons at once **might** work)

Alias

- Alias means: an **alternative name**
- You can have alternative names for commands
- The `alias` shell built-in command allows you to manage aliases
- Now, why have different names to begin with? Isn't it hard enough to remember the official name!?
- Well, actually aliases can be quite useful, time-saving tools in the system...

Alias

- Each **`alias`** line has the format:

```
alias ll='ls -l'
```

- That is: **`alias x='y'`** (with **`x`** being **`ll`** and **`y`** being **`ls -l`** in this case)

- The left side is what the alias will be (the new “command name”), and the right side is the complete command it stands for

- You can create any alias you want using the **`alias`** command. Try:

```
alias lla='ls -l -a'
```

Remember...

```
type ls
```

Let's look at the contents of the root directory of the **remote server**, with details:

```
ls -l /
```

Feeling lazy? Type...

```
ll /
```

Alias

`ll`

is an alias for

`ls -l`

So, whenever you want **`ls -l`**, you can just type **`ll`** instead!

But...

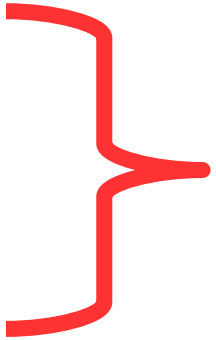
- Your brand new alias will disappear as soon as you log out of the shell
- But if the `alias` command is in the file `.profile` then it will be available every time you log in
- `.profile` is a special file that Bash reads every time you log in
- Another file that contains aliases is the `.bashrc` file
- You will have to edit one of those files to make your alias be there every time you log into your account (we'll learn later how to edit files)

Quiz time!

Go to the course page and choose **Quiz 4**

Kinds of files in Linux

- Regular files (`-`)
- Directories (`d`)
- Links (`l`)
- Character device file (`c`)
- Block device file (`b`)
- Named pipe (`p`)
- Local socket file (`s`)



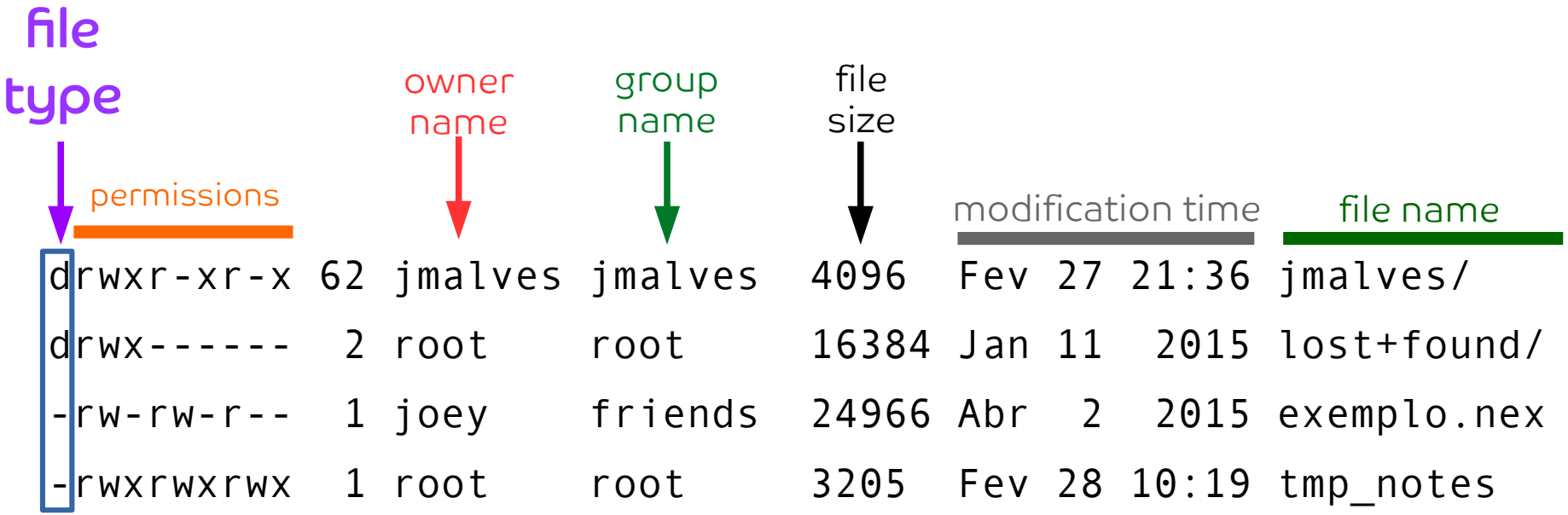
99% of your time as a regular user, you only work with these three kinds of file

See `https://linuxconfig.org/identifying-file-types-in-linux` for more details

File kinds – how to find out?

ls -l

(lists files with details)



File (and system) security

- In Linux, and other Unix-like systems, files have an **owner** and a **group** to which they belong
- Also, each file has **permissions** determined for the owner, the group, and **everyone else**

Kinds of permission

Read

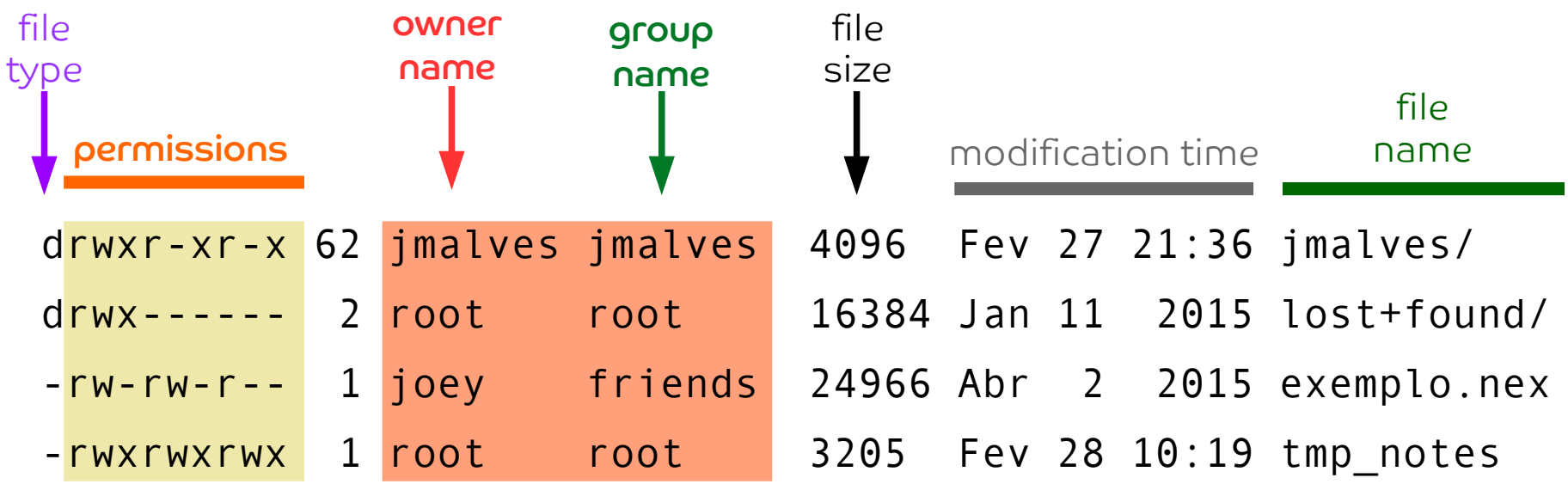
Write

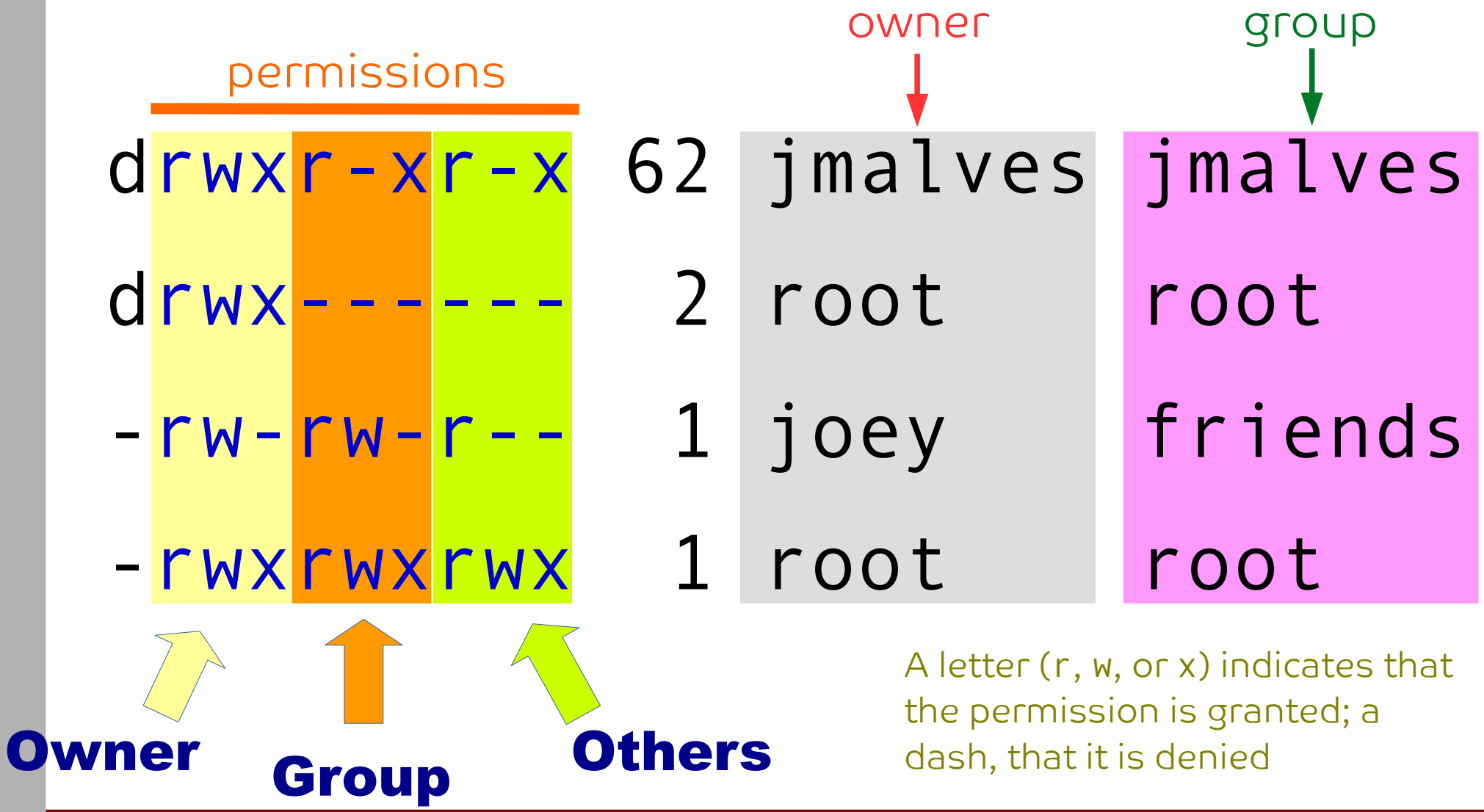
Execute

File ownership and permissions

some examples

```
ls -l
```




permissions
owner
group
drwxr-xr-x 62 jmalves jmalves
drwx------ 2 root root
-rw-rw-r-- 1 joey friends
-rwxrwxrwx 1 root root
Owner
Group
Others
A letter (r, w, or x) indicates that the permission is granted; a dash, that it is denied

Why are users and permissions so important?

- The most obvious reason: To **prevent** (or not) other people from **reading or modifying** your files and directories

- For security: When a program runs, it takes on the effective permissions of the user who started it, and no more (so, for example: **If you cannot** read a certain directory, **a program you start cannot** read it either)

- This also **protects system** files and configurations from regular users

Quiz time!

Go to the course page and choose **Quiz 5**

Groups

- Since Unix systems are time-sharing, with potentially multiple users accessing the same system, it might be interesting to share files or resources **only with certain other users**
- Instead of either limiting access to the user who owns the file or giving it to everyone, there's an intermediate level: the **group**
- A user must belong to **at least one** group (whose name is identical to the user name, usually), but can also belong **to additional groups**
- To see what groups you belong to, use the `groups` command
- Try it in the **remote machine**. What was the output?

Let's try!

- In the **remote machine**, go to directory `~jmalves/PE8/`
- Read the contents of file **`my_file_a`** with:

```
more my_file_a
```

- What was the output?
- Why?
- List this file with details so you can see permissions and ownership info

How about directories?

- Ownership and permissions for a directory work in a **similar** way to those for regular files
- The exception is the **meaning of "execute"** for directories: here, it means `cd` into the directory (i.e., enter the directory)
- Write, for a directory, means modifying it, i.e., creating or removing files
- Read is the same as for files: see what's inside

Let's try!

- In the **remote machine**, inside directory `~jmalves/PE8/`
- Try commands `ls -l` and `cd` on directories:

```
another_dir

also_a_dir

some_dir

third_dir
```

Recap

- The path is the way to get to a file in the system
- The path can be **absolute** (starting from the root directory of the system, or `/`) or **relative** (to the user's current working directory)
- The `.` and `..` abstractions are important representations in the system: current working directory and one directory up in the file tree
- Commands `ls` and `cd` allow us to list file system contents and to move around the structure; `ls -l` lets you see details about the file, such as its type
- Good to know `cd` shortcuts: `cd` by itself takes you home; `cd -` takes you to your immediately previous working directory

Recap

- The `alias` command is a handy way to have shorter commands that are equivalent to much longer ones

- The permission and user/group systems in Unix are **essential and very useful security features**