

# Unix/Linux Induction

or: How I Learned to Stop Worrying and Love the `:(){:|:&;:`

Jascha Schewtschenko

Institute of Cosmology and Gravitation, University of Portsmouth

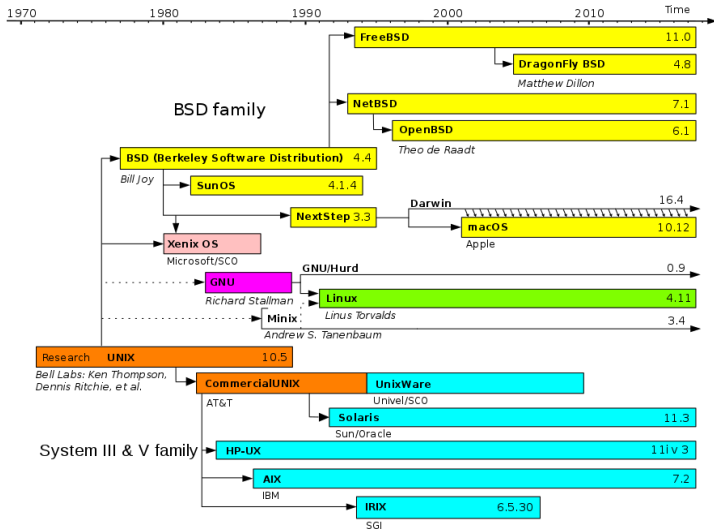
October 9, 2019



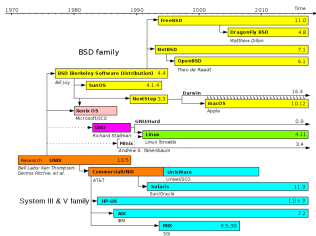
# Outline

- 1 Linux vs Unix vs macOS
- 2 Shells
- 3 Filesystem(s)
- 4 Pipes and input/output control
- 5 Printing
- 6 Software
- 7 Process/Job control
- 8 Scripting, text editing, etc.
- 9 Help/Manpages

# Linux vs Unix vs MacOS

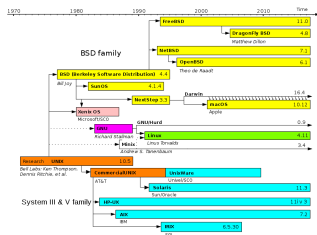


# Linux vs Unix vs MacOS



- Unix Multitasking, multiuser computer operating system (OS); developed in 1970s; modular set of programs/tools; shell scripting to combine tools

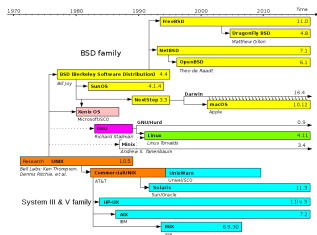
# Linux vs Unix vs MacOS



**Unix** Multitasking, multiuser computer operating system (OS); developed in 1970s; modular set of programs/tools; shell scripting to combine tools

**GNU/Linux** [abbr. L<sub>(in)u</sub>x i<sub>(s)</sub> n<sub>(o)</sub> t u<sub>(n)</sub> i x] ... but close enough; Linux = kernel (core OS) developed by Linus Torvalds in 1991 with the GNU software stack on top (compilers, editors, GUIs, etc). Distros (e.g. Red Hat) bundle software with Linux kernel.

# Linux vs Unix vs MacOS



**Unix** Multitasking, multiuser computer operating system (OS); developed in 1970s; modular set of programs/tools; shell scripting to combine tools

**GNU/Linux** [abbr. L<sub>(inux)</sub>i<sub>(s)</sub>n<sub>(ot)</sub>u<sub>(ni)</sub>x] ... but close enough; Linux = kernel (core OS) developed by Linus Torvalds in 1991 with the GNU software stack on top (compilers, editors, GUIs, etc). Distros (e.g. Red Hat) bundle software with Linux kernel.

**macOS** Unix-based OS developed in early 2000s exclusively for Apple's Macintosh computers (not to be confused with 'classic' Mac OS !);

# Linux vs Unix vs MacOS (cont.)

- Close relationship between OSs makes it possible to port programs from one to another, e.g. macOS supports many of the libraries found in Linux which allows to easily\* compile Linux programs on MacOS (\*adjustments have to be made; reverse portability not that easy)

# Linux vs Unix vs MacOS (cont.)

- Close relationship between OSs makes it possible to port programs from one to another, e.g. macOS supports many of the libraries found in Linux which allows to easily\* compile Linux programs on MacOS (\*adjustments have to be made; reverse portability not that easy)
- Most astrophysics software will work fairly straightforwardly on either OS



# Shells

- A shell is a program/interpreter that works as an interface between the user and the OS via:

# Shells

- A shell is a program/interpreter that works as an interface between the user and the OS via:  
text/command-line interface (CLI) e.g. MS-DOS, sh, csh, bash;  
allows to run commands sequentially (or pre-scripted) to execute tools installed on the OS.

# Shells

- A shell is a program/interpreter that works as an interface between the user and the OS via:
  - text/command-line interface (CLI) e.g. MS-DOS, sh, csh, bash;  
allows to run commands sequentially (or pre-scripted) to execute tools installed on the OS.
  - graphical user interface (GUI) e.g. MS Windows desktop, X window system & window managers/tools; allow users to access data via a menu/shortcut-driven graphical interface with tools allowing for graphical representation/manipulation of e.g. files.

# Shells

- A shell is a program/interpreter that works as an interface between the user and the OS via:
  - text/command-line interface (CLI) e.g. MS-DOS, sh, csh, bash;  
allows to run commands sequentially (or pre-scripted) to execute tools installed on the OS.
  - graphical user interface (GUI) e.g. MS Windows desktop, X window system & window managers/tools; allow users to access data via a menu/shortlink-driven graphical interface with tools allowing for graphical representation/manipulation of e.g. files.
- some OS have a fixed shell (e.g. Windows, macOS), others like Linux/Unix have a wide variety of shells, both graphical (e.g. KDE, Gnome, Unity) as well as CLI (e.g. sh, bash, csh)

# Shells

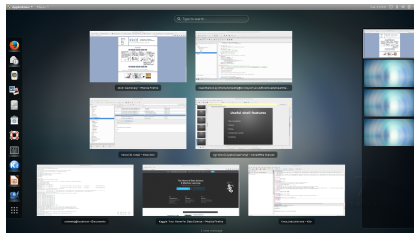
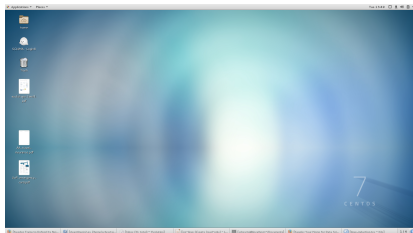
- A shell is a program/interpreter that works as an interface between the user and the OS via:
  - text/command-line interface (CLI) e.g. MS-DOS, sh, csh, bash;  
allows to run commands sequentially (or pre-scripted) to execute tools installed on the OS.
  - graphical user interface (GUI) e.g. MS Windows desktop, X window system & window managers/tools; allow users to access data via a menu/shortlink-driven graphical interface with tools allowing for graphical representation/manipulation of e.g. files.
- some OS have a fixed shell (e.g. Windows, macOS), others like Linux/Unix have a wide variety of shells, both graphical (e.g. KDE, Gnome, Unity) as well as CLI (e.g. sh, bash, csh)
- In this course, we will focus on GNOME3 and bash, as they are pre-installed shells on our (newer) Centos7 Linux distro (tutorials on other shells can be found online)

# Graphical shell: GNOME

- GNOME project is part of GNU project; open source; developed by volunteers and paid contributors since 1999

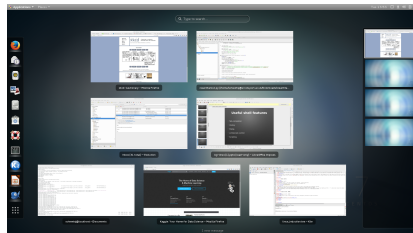
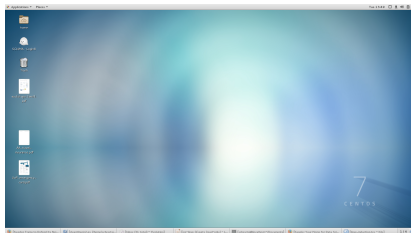
# Graphical shell: GNOME

- GNOME project is part of GNU project; open source; developed by volunteers and paid contributors since 1999
- Layout (Desktop/Overview mode):



# Graphical shell: GNOME

- GNOME project is part of GNU project; open source; developed by volunteers and paid contributors since 1999
- Layout (Desktop/Overview mode):



- useful features:
  - online services** many tools tie in with cloud services like Google Drive or DropBox
  - mouse-buffer** Mark text anywhere and insert this text anywhere else by clicking onto the middle mouse button (or alternatively, left and right button)



# Command-line shell: bash

- **b**(ourne)**a**(gain)**sh**(ell) - default shell in most systems nowadays

## Command-line shell: bash

- **b**(ourne)**a**(gain)**sh**(ell) - default shell in most systems nowadays
- within the graphical environment, you can get access to a command-line shell by opening a terminal e.g. the default gnome-terminal

# Command-line shell: bash

- **b**(ourne)**a**(gain)**sh**(ell) - default shell in most systems nowadays
- within the graphical environment, you can get access to a command-line shell by opening a terminal e.g. the default gnome-terminal
- useful features:

# Command-line shell: bash

- **b**(ourne)**a**(gain)**sh**(ell) - default shell in most systems nowadays
- within the graphical environment, you can get access to a command-line shell by opening a terminal e.g. the default gnome-terminal
- useful features:

**tab-completion** complete commands and filenames by using TAB key

**history** list of previously executed commands (or cycle through them)

**piping** directly using the output of one tool as input for another, e.g. `ls -l | grep test`

**job control** halt/continue/kill/renice processes

**scripting** execute multiple commands, check for conditions (bash actually touring-complete)

# Command-line shell: bash

- **b(ourne)a(gain)sh(ell)** - default shell in most systems nowadays
- within the graphical environment, you can get access to a command-line shell by opening a terminal e.g. the default gnome-terminal
- useful features:
  - tab-completion** complete commands and filenames by using TAB key
  - history** list of previously executed commands (or cycle through them)
  - piping** directly using the output of one tool as input for another, e.g. `ls -l | grep test`
  - job control** halt/continue/kill/renice processes
  - scripting** execute multiple commands, check for conditions (bash actually *touring-complete*)
- We will have a look at useful built-in commands and shell scripting a bit later.

## Remote shell

- **S**(ecure)**SH**(ell) - main way to access Linux remotely - safe and efficient (alternatives like rlogin/rsh should not be used as they transmit their data unencrypted)

# Remote shell

- **S**(ecure)**SH**(ell) - main way to access Linux remotely - safe and efficient (alternatives like rlogin/rsh should not be used as they transmit their data unencrypted)
- In Unix/Linux/macOS, you can simply use it from the command line:

```
$ ssh -Y <username>@<remote machine address>
```

For Windows, you will have to install an ssh client (e.g. PuTTY), which comes with a graphical interface

# Remote shell

- **S**(ecure)**SH**(ell) - main way to access Linux remotely - safe and efficient (alternatives like rlogin/rsh should not be used as they transmit their data unencrypted)
- In Unix/Linux/macOS, you can simply use it from the command line:

```
$ ssh -Y <username>@<remote machine address>
```

For Windows, you will have to install an ssh client (e.g. PuTTY), which comes with a graphical interface

- The actual shell, you work with then, will be the shell on the remote system (e.g. bash)



# Remote shell

- **S**(ecure)**SH**(ell) - main way to access Linux remotely - safe and efficient (alternatives like rlogin/rsh should not be used as they transmit their data unencrypted)
- In Unix/Linux/macOS, you can simply use it from the command line:

```
$ ssh -Y <username>@<remote machine address>
```

For Windows, you will have to install an ssh client (e.g. PuTTY), which comes with a graphical interface

- The actual shell, you work with then, will be the shell on the remote system (e.g. bash)
- If you have an X server installed (Linux by default, Windows/Xming, macOS/XQuartz), ssh also allows graphical tools to be “X-forwarded” onto your local desktop with the “-Y” option when running the command.

# Remote shell

- **S**(ecure)**SH**(ell) - main way to access Linux remotely - safe and efficient (alternatives like rlogin/rsh should not be used as they transmit their data unencrypted)
- In Unix/Linux/macOS, you can simply use it from the command line:

```
$ ssh -Y <username>@<remote machine address>
```

For Windows, you will have to install an ssh client (e.g. PuTTY), which comes with a graphical interface

- The actual shell, you work with then, will be the shell on the remote system (e.g. bash)
- If you have an X server installed (Linux by default, Windows/Xming, macOS/XQuartz), ssh also allows graphical tools to be “X-forwarded” onto your local desktop with the “-Y” option when running the command.
- Alternatively, you can use remote desktop software like X2Go, to get a full remote graphical shell.

# Filesystem(s)

- Manages the storage space on a device; allows you to create directories and files to store/access/organize our data

# Filesystem(s)

- Manages the storage space on a device; allows you to create directories and files to store/access/organize our data
- Various filesystems exist/are used throughout the institute: e.g. ext4 (Linux), HFS+ (macOS), NTFS (Windows), NFS & Lustre (network/SCIAMA)

# Filesystem(s)

- Manages the storage space on a device; allows you to create directories and files to store/access/organize our data
- Various filesystems exist/are used throughout the institute: e.g. ext4 (Linux), HFS+ (macOS), NTFS (Windows), NFS & Lustre (network/SCIAMA)
- if you connect a device formatted for a specific filesystem, your OS has to support it in order to access it

# Filesystem(s): Layouts

- Unix-like filesystems have a similar standard layout to organize their files (and even other resources like devices) within the filesystem:

# Filesystem(s): Layouts

- Unix-like filesystems have a similar standard layout to organize their files (and even other resources like devices) within the filesystem:
  - / This is the *root*. Everything is stored in this top-level directory

# Filesystem(s): Layouts

- Unix-like filesystems have a similar standard layout to organize their files (and even other resources like devices) within the filesystem:
  - / This is the *root*. Everything is stored in this top-level directory
  - /bin/ Contains common binaries (see also /usr/bin)



# Filesystem(s): Layouts

- Unix-like filesystems have a similar standard layout to organize their files (and even other resources like devices) within the filesystem:
  - / This is the *root*. Everything is stored in this top-level directory
  - /bin/ Contains common binaries (see also /usr/bin)
  - /var/ Variable data such as log files

# Filesystem(s): Layouts

- Unix-like filesystems have a similar standard layout to organize their files (and even other resources like devices) within the filesystem:
  - / This is the *root*. Everything is stored in this top-level directory
  - /bin/ Contains common binaries (see also /usr/bin)
  - /var/ Variable data such as log files
  - /usr/ Contains binaries, libraries or shared files for installed user programs

# Filesystem(s): Layouts

- Unix-like filesystems have a similar standard layout to organize their files (and even other resources like devices) within the filesystem:
  - / This is the *root*. Everything is stored in this top-level directory
  - /bin/ Contains common binaries (see also /usr/bin)
  - /var/ Variable data such as log files
  - /usr/ Contains binaries, libraries or shared files for installed user programs
  - /mnt/ Mount points for e.g. external storage devices (see also /media/)

# Filesystem(s): Layouts

- Unix-like filesystems have a similar standard layout to organize their files (and even other resources like devices) within the filesystem:
  - / This is the *root*. Everything is stored in this top-level directory
  - /bin/ Contains common binaries (see also /usr/bin)
  - /var/ Variable data such as log files
  - /usr/ Contains binaries, libraries or shared files for installed user programs
  - /mnt/ Mount points for e.g. external storage devices (see also /media/)
  - /tmp/ Temporary data

## Filesystem(s): Layouts (cont.)

- Some locations may vary depending on the OS, e.g. for the users' home directories:
  - `/home/` common location on Linux (on our systems, it's actually `/home/UNI/<username>` for your network-based home directories)
  - `/Users/` location on macOS
  - `/users/` location on SCIAMA

## Filesystem(s): Layouts (cont.)

- Some locations may vary depending on the OS, e.g. for the users' home directories:
  - `/home/` common location on Linux (on our systems, it's actually `/home/UNI/<username>` for your network-based home directories)
  - `/Users/` location on macOS
  - `/users/` location on SCIAMA
- There are also some shortcuts defined:
  - `.` points to same directory
  - `..` points to parent directory
  - `~` location of your home directory
  - `~<username>` location of the home homedirectory of user `<username>`

# Filesystem(s): File paths

- A path to a file or directory can be given as an *absolute* or a *relative path*

# Filesystem(s): File paths

- A path to a file or directory can be given as an *absolute* or a *relative path*
- an absolute path always starts at the root, e.g.  
`/home/juser/Documents/test.dat`



## Filesystem(s): File paths

- A path to a file or directory can be given as an *absolute* or a *relative path*
- an absolute path always starts at the root, e.g.  
`/home/juser/Documents/test.dat`
- a relative path is given with the current directory as its base, e.g. if the current directory is `/home/juser`, then the relative path to the same data file given above would be simply `Documents/test.dat`, but also alternatively `./Documents/test.dat` or even `../../../../etc/../../../../home/juser/Documents/test.dat`

## Filesystem(s): File paths with regex/escaping

- One or multiple files can be selected by using certain wildcards that your shell understands and expands, e.g.:

## Filesystem(s): File paths with regex/escaping

- One or multiple files can be selected by using certain wildcards that your shell understands and expands, e.g.:
  - \* matches anything
  - ? matches exactly one single character
  - [<set of characters> ] matches one instance of any of the listed characters

## Filesystem(s): File paths with regex/escaping

- One or multiple files can be selected by using certain wildcards that your shell understands and expands, e.g.:

- \* matches anything

- ? matches exactly one single character

- [<set of characters> ] matches one instance of any of the listed characters

e.g. `'?[a,d][0-3]*'` matches `'aa3blub'`, but not `'aA3blub'`, `'ac3ee'` or `'ad4'`

- You can (but should not!) have such special characters as part of your file/directory name. To tell the shell to not treat them as wildcards, whitespace, etc., you have to explicitly 'escape' them, e.g. to select the file with the name `'sh*tty filename?'`, you would have to write it as `'sh\*tty\ filename\?'`

## Filesystem(s): File paths with regex/escaping

- One or multiple files can be selected by using certain wildcards that your shell understands and expands, e.g.:

- \* matches anything

- ? matches exactly one single character

- [<set of characters> ] matches one instance of any of the listed characters

e.g. `'?[a,d][0-3]*'` matches `'aa3blub'`, but not `'aA3blub'`, `'ac3ee'` or `'ad4'`

- You can (but should not!) have such special characters as part of your file/directory name. To tell the shell to not treat them as wildcards, whitespace, etc., you have to explicitly 'escape' them, e.g. to select the file with the name `'sh*tty filename?'`, you would have to write it as `'sh\*tty\ filename\?'`
- alternatively, you can use quote marks i.e. `'sh*tty filename?'`

# Filesystem(s): Management

- When working on the CLI, there are various built-in commands, you can use to navigate and manipulate the filesystem (again, those are identical or very similar on most UNIX-based shells):

# Filesystem(s): Management

- When working on the CLI, there are various built-in commands, you can use to navigate and manipulate the filesystem (again, those are identical or very similar on most UNIX-based shells):
  - `cd <path>` changes current directory to that given by abs. or rel. path (if none is given, it changes into home dir)

# Filesystem(s): Management

- When working on the CLI, there are various built-in commands, you can use to navigate and manipulate the filesystem (again, those are identical or very similar on most UNIX-based shells):
  - `cd <path>` changes current directory to that given by abs. or rel. path (if none is given, it changes into home dir)
  - `ls <path>` lists files at given path (by default current dir)



# Filesystem(s): Management

- When working on the CLI, there are various built-in commands, you can use to navigate and manipulate the filesystem (again, those are identical or very similar on most UNIX-based shells):
  - `cd <path>` changes current directory to that given by abs. or rel. path (if none is given, it changes into home dir)
  - `ls <path>` lists files at given path (by default current dir)
  - `mv <src path> <target path>` moves file(s) given by source path to target path (if target is directory, files keep their names and are moved into it)

# Filesystem(s): Management

- When working on the CLI, there are various built-in commands, you can use to navigate and manipulate the filesystem (again, those are identical or very similar on most UNIX-based shells):
  - `cd <path>` changes current directory to that given by abs. or rel. path (if none is given, it changes into home dir)
  - `ls <path>` lists files at given path (by default current dir)
  - `mv <src path> <target path>` moves file(s) given by source path to target path (if target is directory, files keep their names and are moved into it)
  - `cp <src path> <target path>` same as in `mv`, but keeps source file

# Filesystem(s): Management

- When working on the CLI, there are various built-in commands, you can use to navigate and manipulate the filesystem (again, those are identical or very similar on most UNIX-based shells):
  - `cd <path>` changes current directory to that given by abs. or rel. path (if none is given, it changes into home dir)
  - `ls <path>` lists files at given path (by default current dir)
  - `mv <src path> <target path>` moves file(s) given by source path to target path (if target is directory, files keep their names and are moved into it)
  - `cp <src path> <target path>` same as in `mv`, but keeps source file
  - `rm <path>` removes file(s) given by path (if target is directory, you will have to use `'-r'` to remove everything in it recursively)

# Filesystem(s): Management

- When working on the CLI, there are various built-in commands, you can use to navigate and manipulate the filesystem (again, those are identical or very similar on most UNIX-based shells):
  - `cd <path>` changes current directory to that given by abs. or rel. path (if none is given, it changes into home dir)
  - `ls <path>` lists files at given path (by default current dir)
  - `mv <src path> <target path>` moves file(s) given by source path to target path (if target is directory, files keep their names and are moved into it)
  - `cp <src path> <target path>` same as in `mv`, but keeps source file
  - `rm <path>` removes file(s) given by path (if target is directory, you will have to use `'-r'` to remove everything in it recursively)
  - `mkdir, rmdir` creates/removes (empty) directories (if not empty, use `rm` instead)

## Filesystem(s): Management (cont.)

- There is a very powerful tool to search for folders/files called `find`

`find <options> <starting path> <expression>`

e.g. to find all `.jpg` files in the `/home` and sub-directories.

`find /home -name *.jpg`

- Another useful tool is `file`, which analyses files and tries to determine the type of their content

`file <path to file/folder>`

- There are also tools to give you informations about used/available disk space:

`df` tells you how much space is left on the disks

`du <path>` tells you how much space is used by given file/directory

## Filesystem(s): Management (cont.)

- There is a very powerful tool to search for folders/files called `find`

`find <options> <starting path> <expression>`

e.g. to find all `.jpg` files in the `/home` and sub-directories.

`find /home -name *.jpg`

- Another useful tool is `file`, which analyses files and tries to determine the type of their content

`file <path to file/folder>`

- There are also tools to give you informations about used/available disk space:

`df` tells you how much space is left on the disks

`du <path>` tells you how much space is used by given file/directory

- results are given in bytes; add `'-h'` argument to get “human-readable” numbers with SI-prefixes

# Filesystem(s): Hidden files

- any file with a leading '.' is treated as a hidden file, i.e. it won't show up in the default listing of `ls`

# Filesystem(s): Hidden files

- any file with a leading '.' is treated as a hidden file, i.e. it won't show up in the default listing of `ls`
- this includes many config files in your home directory



# Filesystem(s): Hidden files

- any file with a leading '.' is treated as a hidden file, i.e. it won't show up in the default listing of `ls`
- this includes many config files in your home directory
- if you want to list them, you have to use `ls` with the `-a` argument, or configure your graphical file manager to also show hidden files

# Filesystem(s): (Hidden) Files - Exercise

## Listing files

- 1 try to list all files/folder, both normal and hidden in your home directory using the `ls` command

# Filesystem(s): Ownership/Permissions

```
[jschewts@login5(sciama) ~]$ ls -l test.dat  
-rwxr---x 1 jschewts users 0 Oct 10 19:39 test.dat
```

# Filesystem(s): Ownership/Permissions

```
[jschewts@login5(sciama) ~]$ ls -l test.dat  
-rwxr---x 1 jschewts users 0 Oct 10 19:39 test.dat
```

user

- each file/directory belongs to a user and group that is listed with e.g.  
`ls -l`

# Filesystem(s): Ownership/Permissions

```
[jschewts@login5(sciama) ~]$ ls -l test.dat  
-rwxr---x 1 jschewts users 0 0ct 10 19:39 test.dat
```

                    user    group

- each file/directory belongs to a user and group that is listed with e.g. `ls -l`
- every user also belongs to one or multiple groups (use `id <username>` to list them)

# Filesystem(s): Ownership/Permissions

```
[ischewts@login5(sciama) ~]$ ls -l test.dat  
-rwxr--r-x 1 ischewts users 0 Oct 10 19:39 test.dat  
  other  user  group
```

- each file/directory belongs to a user and group that is listed with e.g. `ls -l`
- every user also belongs to one or multiple groups (use `id <username>` to list them)
- the filesystem distinguishes between three different permission types for each file/directory:
  - `r(ead)` allows to read content of file / to list content of directory
  - `w(rite)` allows to change content of file / to manipulate file list of directory (i.e. create, remove, rename files)
  - `(e)x(ecute)` allows to execute file / to enter directory

# Filesystem(s): Ownership/Permissions

```
[jischewts@login5(sciama) ~]$ ls -l test.dat
-rwxr--r-x 1 jischewts users 0 0ct 10 19:39 test.dat
  other  user  group
```

- ownerships/permissions can be changed using the commands `chown`, `chgrp` and `chmod` respectively

# Filesystem(s): Ownership/Permissions

```
[jischewts@login5(sciama) ~]$ ls -l test.dat  
-rwxr--r-x 1 jischewts users 0 Oct 10 19:39 test.dat  
  other  user  group
```

- ownerships/permissions can be changed using the commands `chown`, `chgrp` and `chmod` respectively
- for changing permissions, you can either add/remove permissions using a string e.g. `chmod g+rw <filename>` for adding read/write permission to the file for the group who owns it



# Filesystem(s): Ownership/Permissions

```
[jischewts@login5(sciama) ~]$ ls -l test.dat  
-rwxr--r-x 1 jischewts users 0 Oct 10 19:39 test.dat
```

other user group

- ownerships/permissions can be changed using the commands `chown`, `chgrp` and `chmod` respectively
- for changing permissions, you can either add/remove permissions using a string e.g. `chmod g+rw <filename>` for adding read/write permission to the file for the group who owns it
- alternative you can set the permissions for user/group/others using a numeral expression based on a bit mask ( $r=4, w=2, x=1$ ): so set the permissions for 'test.dat' as shown, you would have to use `chmod 741 test.dat`

# Filesystem(s): Ownership/Permissions

```
[ischewts@login5(sciama) ~]$ ls -l test.dat  
-rwxr--r-x 1 ischewts users 0 Oct 10 19:39 test.dat
```

other user group

- ownerships/permissions can be changed using the commands `chown`, `chgrp` and `chmod` respectively
- for changing permissions, you can either add/remove permissions using a string e.g. `chmod g+rw <filename>` for adding read/write permission to the file for the group who owns it
- alternative you can set the permissions for user/group/others using a numeral expression based on a bit mask ( $r=4, w=2, x=1$ ): so set the permissions for 'test.dat' as shown, you would have to use `chmod 741 test.dat`
- all three commands support the argument `'-R'` for a directory which changes the ownership/permission for all files/dirs in it recursively

# Filesystem(s): Ownership/Permissions - Exercise

## Changing permissions and ownership

- 1 create an empty file using

```
touch test.dat
```

- 2 Check its current permissions and ownership
- 3 Now remove all permissions for anyone but the owner.
- 4 Try to open the file e.g. with text editor `gedit` and edit it and save the changes. Now remove the write permission and try to edit/save it again.
- 5 Now close the editor, change the ownership of that file to e.g. 'root' and try to open it again.

# Pipes and input/output control

- standard streams

**STDIN** is usually your keyboard input, but be also redirected output from other devices/a file/other program

**STDOUT** usually output shown directly in the terminal, but can be (re)directed into a file/device/program

**STDERR** second type of output used for important status/error message usually output shown directly in the terminal (can also be redirected)

# Pipes and input/output control

- standard streams

**STDIN** is usually your keyboard input, but be also redirected output from other devices/a file/other program

**STDOUT** usually output shown directly in the terminal, but can be (re)directed into a file/device/program

**STDERR** second type of output used for important status/error message usually output shown directly in the terminal (can also be redirected)

- ' | ' creates a pipe between the STDOUT of one program and the STDIN of another, e.g. `ls | grep blub`

# Pipes and input/output control

- standard streams

**STDIN** is usually your keyboard input, but be also redirected output from other devices/a file/other program

**STDOUT** usually output shown directly in the terminal, but can be (re)directed into a file/device/program

**STDERR** second type of output used for important status/error message usually output shown directly in the terminal (can also be redirected)

- '`|`' creates a pipe between the **STDOUT** of one program and the **STDIN** of another, e.g. `ls | grep blub`
- '`> <filename>`' can be used to direct the **STDOUT** into the specified file (will overwrite existing content, for appending use '`>>`' instead), e.g. `./my_program.sh > output.dat`

# Pipes and input/output control

- standard streams

**STDIN** is usually your keyboard input, but be also redirected output from other devices/a file/other program

**STDOUT** usually output shown directly in the terminal, but can be (re)directed into a file/device/program

**STDERR** second type of output used for important status/error message usually output shown directly in the terminal (can also be redirected)

- '`|`' creates a pipe between the **STDOUT** of one program and the **STDIN** of another, e.g. `ls | grep blub`
- '`> <filename>`' can be used to direct the **STDOUT** into the specified file (will overwrite existing content, for appending use '`>>`' instead), e.g. `./my_program.sh > output.dat`
- '`< <filename>`' can be used to use the content of a file for **STDIN**, e.g. `qsub < submission.batch`

# Printing

- ICG printers (ICGcolor, ICGbw, ICGsupport) should be pre-configured on our Linux Desktops/Laptops



# Printing

- ICG printers (ICGcolor, ICGbw, ICGsupport) should be pre-configured on our Linux Desktops/Laptops
- `lpr -P <printer name> <filename>` can directly print postscript/ascii files from the CLI

# Printing

- ICG printers (ICGcolor, ICGbw, ICGsupport) should be pre-configured on our Linux Desktops/Laptops
- `lpr -P <printer name> <filename>` can directly print postscript/ascii files from the CLI
- `lpq -P <printer name>` lists all jobs in the printer queue

# Printing

- ICG printers (ICGcolor, ICGbw, ICGsupport) should be pre-configured on our Linux Desktops/Laptops
- `lpr -P <printer name> <filename>` can directly print postscript/ascii files from the CLI
- `lpq -P <printer name>` lists all jobs in the printer queue
- `lprm -P <printer name> <job nr>` removes specific job from the printer queue

# Printing

- ICG printers (ICGcolor, ICGbw, ICGsupport) should be pre-configured on our Linux Desktops/Laptops
- `lpr -P <printer name> <filename>` can directly print postscript/ascii files from the CLI
- `lpq -P <printer name>` lists all jobs in the printer queue
- `lprm -P <printer name> <job nr>` removes specific job from the printer queue
- you can always print from graphical programs (Acrobat, GhostView, text editors, etc.)

# Software

- Software may be installed from managed package trees using tools like e.g. aptitude or yum on Linux (or graphical programs like Synaptic) or e.g. homebrew on macOS

# Software

- Software may be installed from managed package trees using tools like e.g. aptitude or yum on Linux (or graphical programs like Synaptic) or e.g. homebrew on macOS
- usually requires administrator rights, which you may have (or obtain with `sudo`) on your own private Linux, but won't have on the managed workstations/laptops in the institute (ask icg-computing if you need any software installed)

# Software

- Software may be installed from managed package trees using tools like e.g. aptitude or yum on Linux (or graphical programs like Synaptic) or e.g. homebrew on macOS
- usually requires administrator rights, which you may have (or obtain with `sudo`) on your own private Linux, but won't have on the managed workstations/laptops in the institute (ask icg-computing if you need any software installed)
- Alternatively, software/libraries can be compiled from source code.

# Software: How to build & run it

- 1 Obtaining source code
- 2 Compiling/Installing software
- 3 Running software



# Software: Download/File compression

- You can download software directly from the internet using the command line by either accessing a version control repository (see below) or using e.g. `wget`:

```
wget <URL to source package>
```

- when dealing with downloaded source code, you usually will encounter it in form of a (gzip-compressed) tar (`*.tar`, `*.tgz`, `*.tar.gz`, `*.tar.bz2`) archive

# Software: Download/File compression

- You can download software directly from the internet using the command line by either accessing a version control repository (see below) or using e.g. `wget`:

```
wget <URL to source package>
```

- when dealing with downloaded source code, you usually will encounter it in form of a (gzip-compressed) tar (`*.tar`, `*.tgz`, `*.tar.gz`, `*.tar.bz2`) archive
- you can create such a compressed archive yourself using `tar cvzf <archive filename> <list of files to be included>`

# Software: Download/File compression

- You can download software directly from the internet using the command line by either accessing a version control repository (see below) or using e.g. `wget`:

```
wget <URL to source package>
```

- when dealing with downloaded source code, you usually will encounter it in form of a (gzip-compressed) tar (`*.tar`, `*.tgz`, `*.tar.gz`, `*.tar.bz2`) archive
- you can create such a compressed archive yourself using `tar cvzf <archive filename> <list of files to be included>`
- to uncompress/unpack a gzip-compressed tar archive, use `tar xvfz <archive filename> <list of files to be included>` (if bzip2-compressed, replace `z` with `j`, or omit it for uncompressed archives)

# Software: Download/File compression - Exercise

## Source from website

- 1 Download the CLASS source code directly from the website `http://class-code.net` using `wget` (Tipp: Obtain the direct URL from right-clicking on the link and copying the 'Link Location':

```
wget http://lesgourg...
```

- 2 Unpack the compressed tar archive

# Software: Source/Version control

- lots of choices (cvs, s(ub)v(ersio)n, etc.) for version control, but nowadays mostly git used

# Software: Source/Version control

- lots of choices (cvs, s(ub)v(ersio)n, etc.) for version control, but nowadays mostly git used
- lets you to keep track of changes to files (e.g. source code, tex documents) to e.g. compare or revert them, use branches/forks

# Software: Source/Version control

- lots of choices (cvs, s(ub)v(ersio)n, etc.) for version control, but nowadays mostly git used
- lets you to keep track of changes to files (e.g. source code, tex documents) to e.g. compare or revert them, use branches/forks
- if remote repository is used (e.g. on github or bitbucket servers), great way to keep backups and share/collaborate with other programmers/writers/users

# Software: Source/Version control

- lots of choices (cvs, s(ub)v(ersio)n, etc.) for version control, but nowadays mostly git used
- lets you to keep track of changes to files (e.g. source code, tex documents) to e.g. compare or revert them, use branches/forks
- if remote repository is used (e.g. on github or bitbucket servers), great way to keep backups and share/collaborate with other programmers/writers/users
- for more details: see data language lecture



# Software: Source/Version control

- lots of choices (cvs, s(ub)v(ersio)n, etc.) for version control, but nowadays mostly git used
- lets you to keep track of changes to files (e.g. source code, tex documents) to e.g. compare or revert them, use branches/forks
- if remote repository is used (e.g. on github or bitbucket servers), great way to keep backups and share/collaborate with other programmers/writers/users
- for more details: see data language lecture
- if you want to obtain the most recent version of the source code for a program, you can often find it on such a server. To get a copy e.g. from a git repository, you would simply call:

```
git clone <URL to repository>
```

# Software: Source/Version control - Exercise

## Source from git

Now try to obtain the CLASS source directly from the git repository (Tipp: You can find the link to the repository on the website and then the url to the repository by clicking the green 'Clone or download' button)

```
git clone ...
```

# Software: Compiling/Installation

- Most software distributed in source code provide Makefiles or supports automake to automatically create optimized Makefiles

# Software: Compiling/Installation

- Most software distributed in source code provide Makefiles or supports automake to automatically create optimized Makefiles
- In the latter case, change into the directory where the source code is located and call `./configure` to trigger the generation/configuration process for the Makefile (usually, you can define the installation path with the additional argument `--prefix=<installation path>`)

# Software: Compiling/Installation

- Most software distributed in source code provide Makefiles or supports automake to automatically create optimized Makefiles
- In the latter case, change into the directory where the source code is located and call `./configure` to trigger the generation/configuration process for the Makefile (usually, you can define the installation path with the additional argument `--prefix=<installation path>`)
- once you have a Makefile (or in case it was already shipped with the source), run the command `make` on the CLI. It uses the instructions in the Makefile to compile the source code.

# Software: Compiling/Installation

- Most software distributed in source code provide Makefiles or supports automake to automatically create optimized Makefiles
- In the latter case, change into the directory where the source code is located and call `./configure` to trigger the generation/configuration process for the Makefile (usually, you can define the installation path with the additional argument `--prefix=<installation path>`)
- once you have a Makefile (or in case it was already shipped with the source), run the command `make` on the CLI. It uses the instructions in the Makefile to compile the source code.
- Install the software: `make install` (this may need administrator rights depending on the installation path, by default system directories)

# Software: Compiling/Installation

- Most software distributed in source code provide Makefiles or supports automake to automatically create optimized Makefiles
- In the latter case, change into the directory where the source code is located and call `./configure` to trigger the generation/configuration process for the Makefile (usually, you can define the installation path with the additional argument `--prefix=<installation path>`)
- once you have a Makefile (or in case it was already shipped with the source), run the command `make` on the CLI. It uses the instructions in the Makefile to compile the source code.
- Install the software: `make install` (this may need administrator rights depending on the installation path, by default system directories)
- In any case, check the `README` or `INSTALL` file shipped with the source for further information on how to compile/install the software

# Software: Compiling/Installation - Exercise

## Compiling CLASS

Check the README file for instructions on how to compile the code and follow them. You should end up with a class binary file. Test the binary using

```
./class explanatory.ini
```



# Software: Environment variables

- Many programs make use of so-called *environment variables* for configuration and/or control

# Software: Environment variables

- Many programs make use of so-called *environment variables* for configuration and/or control
- Important ones are e.g. `HOME` (pointing to your home dir), `PATH` (used by the shell to find binaries) or `LD_LIBRARY_PATH` (used by the linker to find shared libraries)

# Software: Environment variables

- Many programs make use of so-called *environment variables* for configuration and/or control
- Important ones are e.g. `HOME` (pointing to your home dir), `PATH` (used by the shell to find binaries) or `LD_LIBRARY_PATH` (used by the linker to find shared libraries)
- In bash, you can list them using `env`, set them using `export <variable name>=<value>` and reference to them by `${<variable name>}`, e.g.

```
export PATH=$HOME/bin:$PATH
```

# Software: Environment variables

- Many programs make use of so-called *environment variables* for configuration and/or control
- Important ones are e.g. HOME (pointing to your home dir), PATH (used by the shell to find binaries) or LD\_LIBRARY\_PATH (used by the linker to find shared libraries)
- In bash, you can list them using `env`, set them using `export <variable name>=<value>` and reference to them by `$<variable name>`, e.g.

```
export PATH=$HOME/bin:$PATH
```

- if you installed your software into a custom directory, you have to make sure that the binaries and libraries are in a directory listed in PATH and (LD\_)LIBRARY\_PATH respectively.

# Software: Environment variables - Exercise

## Adding a program to PATH

Copy the `explanatory.ini` into your home dir

```
cp explanatory.ini /
```

Change now into your home dir

```
cd
```

In order to run `class`, you would now need to use the full path to the binary. But instead, try to add the `CLASS` folder containing the binary to the `PATH` as described above. Now try to run `class` as a simple command

```
class explanatory.ini
```

# Process/Job control: Jobs

- you start a new *process/job* in a shell each time, you execute a command/program.

# Process/Job control: Jobs

- you start a new *process/job* in a shell each time, you execute a command/program.
- *Jobs* are groups of processes started by a single command (e.g. a script or program which starts itself a couple of processes), you can list(and manipulate) them for the current shell with the command `jobs`

## Process/Job control: Jobs

- you start a new *process/job* in a shell each time, you execute a command/program.
- *Jobs* are groups of processes started by a single command (e.g. a script or program which starts itself a couple of processes), you can list(and manipulate) them for the current shell with the command `jobs`
- Jobs are by default running in the *foreground*, blocking the shell until it is finished/terminated (actually catching the `STDIO` of the shell)



# Process/Job control: Jobs

- you start a new *process/job* in a shell each time, you execute a command/program.
- *Jobs* are groups of processes started by a single command (e.g. a script or program which starts itself a couple of processes), you can list(and manipulate) them for the current shell with the command `jobs`
- Jobs are by default running in the *foreground*, blocking the shell until it is finished/terminated (actually catching the `STDIO` of the shell)
- You can make run in the *background* by adding `'&'` at the end of the command, e.g. `xterm &` (obviously do not send programs into the background that need user interactions!)

# Process/Job control: Jobs

- you start a new *process/job* in a shell each time, you execute a command/program.
- *Jobs* are groups of processes started by a single command (e.g. a script or program which starts itself a couple of processes), you can list(and manipulate) them for the current shell with the command `jobs`
- Jobs are by default running in the *foreground*, blocking the shell until it is finished/terminated (actually catching the `STDIO` of the shell)
- You can make run in the *background* by adding `'&'` at the end of the command, e.g. `xterm &` (obviously do not send programs into the background that need user interactions!)
- `fg [<jobnumber>]` will make it come to the foreground again

# Process/Job control: Jobs

- you start a new *process/job* in a shell each time, you execute a command/program.
- *Jobs* are groups of processes started by a single command (e.g. a script or program which starts itself a couple of processes), you can list(and manipulate) them for the current shell with the command `jobs`
- Jobs are by default running in the *foreground*, blocking the shell until it is finished/terminated (actually catching the `STDIO` of the shell)
- You can make run in the *background* by adding `'&'` at the end of the command, e.g. `xterm &` (obviously do not send programs into the background that need user interactions!)
- `fg [<jobnumber>]` will make it come to the foreground again
- `CTRL-z` suspends a foreground job (it will be stopped then and you can use `bg [<jobnumber>]` to make it run in the background)

## Process/Job control: Processes

- you can also interact with processes (but be careful, that may interrupt jobs or vital functions of your OS)

## Process/Job control: Processes

- you can also interact with processes (but be careful, that may interrupt jobs or vital functions of your OS)
- you can get a list of all processes currently running on the system by using `ps` `[aux]` or `top`

## Process/Job control: Processes

- you can also interact with processes (but be careful, that may interrupt jobs or vital functions of your OS)
- you can get a list of all processes currently running on the system by using `ps [aux]` or `top`
- each process has a p(rocess)id as well as a p(arent)pid of the process who spawned it

## Process/Job control: Processes

- you can also interact with processes (but be careful, that may interrupt jobs or vital functions of your OS)
- you can get a list of all processes currently running on the system by using `ps [aux]` or `top`
- each process has a p(rocess)id as well as a p(arent)pid of the process who spawned it
- you can send various signals to processes using the `kill` `[<signal>] <pid>` command:

## Process/Job control: Processes

- you can also interact with processes (but be careful, that may interrupt jobs or vital functions of your OS)
- you can get a list of all processes currently running on the system by using `ps [aux]` or `top`
- each process has a p(rocess)id as well as a p(arent)pid of the process who spawned it
- you can send various signals to processes using the `kill`

[<signal>] <pid> command:

**SIGTERM** (15) (default) - Termination signal i.e. asks running process to terminate itself.

**SIGINT** (2) - Interrupt signal (same as CTRL-c on keyboard; usually same effect as SIGTERM).

**SIGKILL** (9) - Kill signal i.e. kill running process.

**SIGSTOP** (19) - Stop process.

**SIGCONT** (18) - Continue process if stopped.



## Process/Job control: Processes

- you can also interact with processes (but be careful, that may interrupt jobs or vital functions of your OS)
- you can get a list of all processes currently running on the system by using `ps [aux]` or `top`
- each process has a p(rocess)id as well as a p(arent)pid of the process who spawned it
- you can send various signals to processes using the `kill`

[<signal>] <pid> command:

**SIGTERM** (15) (default) - Termination signal i.e. asks running process to terminate itself.

**SIGINT** (2) - Interrupt signal (same as CTRL-c on keyboard; usually same effect as SIGTERM).

**SIGKILL** (9) - Kill signal i.e. kill running process.

**SIGSTOP** (19) - Stop process.

**SIGCONT** (18) - Continue process if stopped.

e.g. `kill -19 1923` stops the process with the pid 1923, while `kill -SIGCONT 1923` will resume it again.

## Process/Job control: Processes (cont.)

- you can also *renice* processes to affect how greedy they are when using the computation time (e.g. a non-urgent non-interactive calculation does not have to be responsive to sudden new input and also does not need to finish as quickly as possible)

## Process/Job control: Processes (cont.)

- you can also *renice* processes to affect how greedy they are when using the computation time (e.g. a non-urgent non-interactive calculation does not have to be responsive to sudden new input and also does not need to finish as quickly as possible)
- `renice <niceness> <pid>` can be used to make jobs “nicer” or “greedier” (the latter usually requires admin rights)

## Process/Job control: Processes (cont.)

- you can also *renice* processes to affect how greedy they are when using the computation time (e.g. a non-urgent non-interactive calculation does not have to be responsive to sudden new input and also does not need to finish as quickly as possible)
- `renice <niceness> <pid>` can be used to make jobs “nicer” or “greedier” (the latter usually requires admin rights)
- Levels of niceness run from -20 to +19, 0 is the default level

# Scripting, text editing, etc.

- Some handy tools

`emacs`, `vim`, `gedit` text editor

`more`, `less` pages through a file

`cat <file> [<file2> ... ]` concatenates files and writes them to  
STDOUT

`head`, `tail` show top/bottom of a file (`tail -f` keeps updating  
bottom, handy e.g. for log files of active program)

`grep <pattern> [<files> ]` parses STDIN or files for pattern (regex)  
and returns matching lines

`sed`, `awk` very powerful CLI stream/text processors; can be used  
to post-process output from a program or quickly  
replace strings in a file (perfect for scripting)

`screen` allows you to detach a shell from the terminal/login  
(e.g. to keep it running while you close the terminal or  
ssh connection and to reattach it to a new session)

## Scripting, text editing, etc. (cont.)

- You can combine multiple tools to build more powerful ones

## Scripting, text editing, etc. (cont.)

- You can combine multiple tools to build more powerful ones
- You can either do this on the command line directly using piping (e.g. `ls *.png | wc -w`) and/or process controls (e.g. `./program1; ./program2 && echo 'success'`)

## Scripting, text editing, etc. (cont.)

- You can combine multiple tools to build more powerful ones
- You can either do this on the command line directly using piping (e.g. `ls *.png | wc -w`) and/or process controls (e.g. `./program1; ./program2 && echo 'success'`)
- or you can put all these command sequences into a file (script) to reuse them (or even combine them again)



## Scripting, text editing, etc. (cont.)

- You can combine multiple tools to build more powerful ones
- You can either do this on the command line directly using piping (e.g. `ls *.png | wc -w`) and/or process controls (e.g. `./program1; ./program2 && echo 'success'`)
- or you can put all these command sequences into a file (script) to reuse them (or even combine them again)
- a script file usually starts with a line like `"#!/usr/bin/env bash"` where you define the interpreter of the script (you can replace "bash" with e.g. `python`, `awk`, etc. if your script should use these programs instead)

## Scripting, text editing, etc. (cont.)

- You can combine multiple tools to build more powerful ones
- You can either do this on the command line directly using piping (e.g. `ls *.png | wc -w`) and/or process controls (e.g. `./program1; ./program2 && echo 'success'`)
- or you can put all these command sequences into a file (script) to reuse them (or even combine them again)
- a script file usually starts with a line like `"#!/usr/bin/env bash"` where you define the interpreter of the script (you can replace "bash" with e.g. python, awk, etc. if your script should use these programs instead)
- do not forget to make your script executable (see file permissions) before trying to run it

## Scripting, text editing, etc. (cont.)

- You can combine multiple tools to build more powerful ones
- You can either do this on the command line directly using piping (e.g. `ls *.png | wc -w`) and/or process controls (e.g. `./program1; ./program2 && echo 'success'`)
- or you can put all these command sequences into a file (script) to reuse them (or even combine them again)
- a script file usually starts with a line like `"#!/usr/bin/env bash"` where you define the interpreter of the script (you can replace "bash" with e.g. python, awk, etc. if your script should use these programs instead)
- do not forget to make your script executable (see file permissions) before trying to run it
- for more information on (shell) scripting, please be referred to the plentiful resources online

# Help/Manpages

- most programs/commands on Linux come with their own man(ual) pages. You can access them using `man <command>`.

# Help/Manpages

- most programs/commands on Linux come with their own man(ual) pages. You can access them using `man <command>`.
- If you are not sure which command to use, you can use `apropos <search term>` to find a command and its manpages.

# Help/Manpages

- most programs/commands on Linux come with their own man(ual) pages. You can access them using `man <command>`.
- If you are not sure which command to use, you can use `apropos <search term>` to find a command and its manpages.
- Google!/Bing!/DuckDuckGo! - Loads of information out there (and if not there are forums like stackoverflow with helpful people)