Introductory Linux Tutorial for Life Sciences

Session 2: The filesystem

Teacher: Marcel Martin

The filesystem

- Conceptually, everything in Unix is a file or a process

The filesystem

- Conceptually, everything in Unix is a file or a process
- A *process* is a running program

The filesystem

- Conceptually, everything in Unix is a file or a process
- A *process* is a running program
- A *file* is a "piece" of data
- Files can be grouped into directories (or: folders)

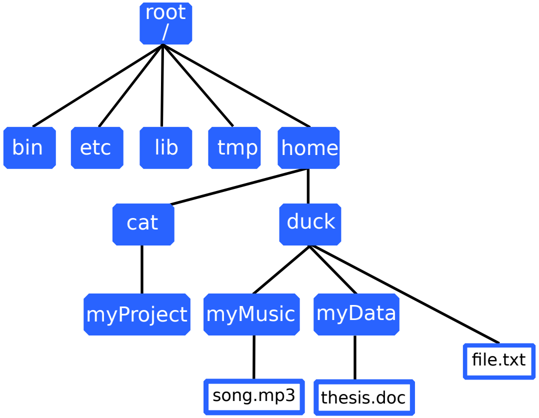
The filesystem

- Conceptually, everything in Unix is a file or a process
- A *process* is a running program
- A *file* is a “piece” of data
- Files can be grouped into directories (or: folders)
- A filesystem is a logical collection of files and directories on a disk managed using a hierarchical filesystem structure

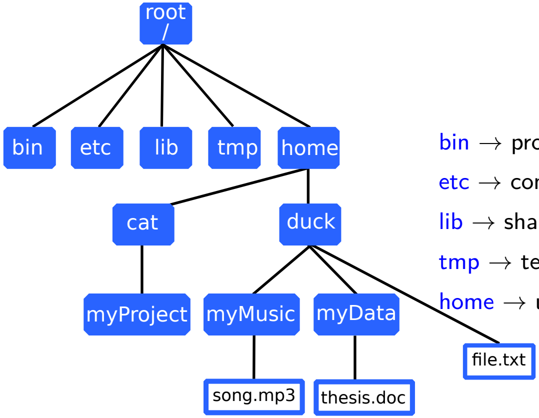
Filesystem properties

- Represented as an upside-down tree with a root directory ('/') at the top
- Each file or directory is uniquely identified by its name
- It is self contained $\rightarrow$ no dependencies between one filesystem and any other

Filesystem structure



Filesystem structure



bin → program files (“binaries”)

etc → configuration files

lib → shared library files

tmp → temporary files storage place

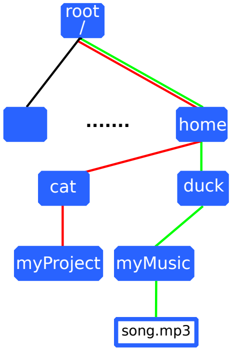
home → users’ home directories

Path

- Location of a file or directory
- A combination of '/' and alpha-numeric characters

  `/home/duck/myMusic/song.mp3`

  `/home/cat/myProject`
- '/' has two meanings:
  - root → in front of a file/directory
  - a separator → inside a path



File and directory names

- Are case sensitive
- Forbidden character: '/'
- Not recommended: [space] [enter] ; * [quotes]

File names

- Often have the form: name.something (`song.mp3`)

File and directory names

- Are case sensitive
- Forbidden character: '/'
- Not recommended: [space] [enter] ; * [quotes]

File names

- Often have the form: name.something (`song.mp3`)
- '.something'= filename extension
- Some extensions are used by convention to indicate the type of data the file holds

File and directory names

- Are case sensitive
- Forbidden character: **'/'**
- Not recommended: [space] [enter] ; * [quotes]

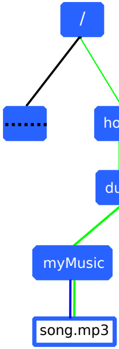
File names

- Often have the form: name.something (`song.mp3`)
- '.something'= filename extension
- Some extensions are used by convention to indicate the type of data the file holds
  - .txt → plain text file
  - .pdf → PDF document
  - .sh, .pl, .py → shell, Perl, Python scripts

Absolute vs relative paths

An *absolute path*

- Looks like `/home/duck/myMusic/song.mp3`
- Starts with '/'
- Describes how to reach a location in the filesystem from the root



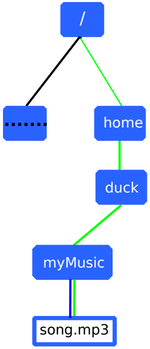
Absolute vs relative paths

An *absolute path*

- Looks like `/home/duck/myMusic/song.mp3`
- Starts with '/'
- Describes how to reach a location in the filesystem from the root

A *relative path*

- Looks like `song.mp3`
- or `myMusic/song.mp3`
- Does not start with '/'
- Describes how to reach a location in the filesystem *from another location*



The “dot” directories

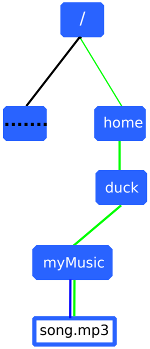
Each directory has two special subdirectories:

- `.` (dot) is the *directory itself*

  `/home/duck/myMusic/./song.mp3`

  is the same as

  `/home/duck/myMusic/song.mp3`



The “dot” directories

Each directory has two special subdirectories:

- `.` (dot) is the *directory itself*

  `/home/duck/myMusic/./song.mp3`

  is the same as

  `/home/duck/myMusic/song.mp3`

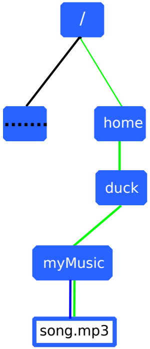
- `..` (dot dot) is the *parent directory*

  `/home/duck/myMusic/..`

  is the same as

  `/home/duck`

The `.` and `..` directories are hidden by default



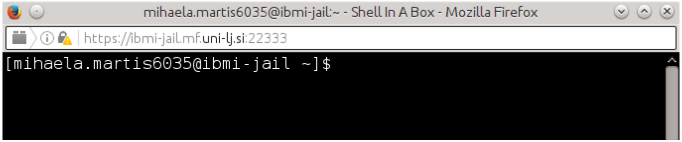
The working directory

- The shell has associated with it[1] a *working directory*
- It marks your "current position" in the filesystem
- Relative paths are relative to the working directory
- Show the working directory with `pwd` (print working directory)
- Change the working directory with `cd` (change directory)

[1] actually: every process

The home directory

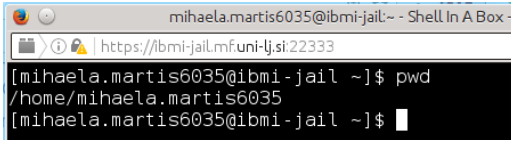
- Your home directory is `/home/username`
- Keep all your data in it
- When you open a new terminal, this is where you start
- That is, the working directory is set to the home directory
- It is abbreviated by the tilde character '~'


mihaela.martis6035@ibmi-jail:~ - Shell In A Box - Mozilla Firefox
https://ibmi-jail.mf.uni-lj.si:22333

```
[mihaela.martis6035@ibmi-jail ~]$
```

Command: `pwd` – print working directory

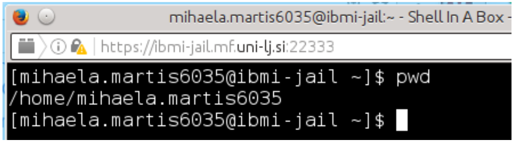
- `pwd` shows the path to the working directory


mihaela.martis6035@ibmi-jail:~ - Shell In A Box -
https://ibmi-jail.mf.uni-lj.si:22333

```
[mihaela.martis6035@ibmi-jail ~]$ pwd
/home/mihaela.martis6035
[mihaela.martis6035@ibmi-jail ~]$ 
```

Command: pwd – print working directory

- pwd shows the path to the working directory


mihaela.martis6035@ibmi-jail:~ - Shell In A Box -
https://ibmi-jail.mf.uni-lj.si:22333

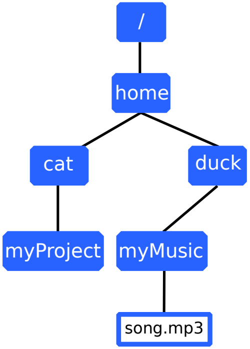
```
[mihaela.martis6035@ibmi-jail ~]$ pwd
/home/mihaela.martis6035
[mihaela.martis6035@ibmi-jail ~]$
```

- pwd is often not needed because the prompt shows the working directory already

Command: cd – change directory

- cd changes the working directory
- Allows you to “move” into another directory. With a relative path:

```
$ cd myMusic
$ pwd
/home/duck/myMusic
```



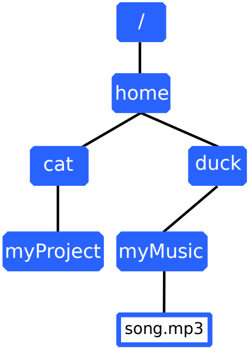
Command: `cd` – change directory

- `cd` changes the working directory
- Allows you to "move" into another directory. With a relative path:

```
$ cd myMusic
$ pwd
/home/duck/myMusic
```

- With an absolute path:

```
$ cd /home/cat/myProject
```



Command: `cd` – change directory

- `cd` changes the working directory
- Allows you to “move” into another directory. With a relative path:

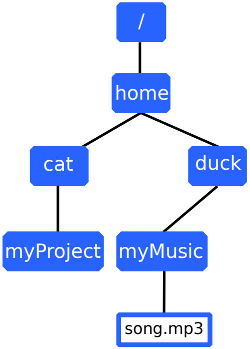
```
$ cd myMusic
$ pwd
/home/duck/myMusic
```

- With an absolute path:

```
$ cd /home/cat/myProject
```

- Go “up” using “`..`”:

```
$ cd ..
$ pwd
/home/cat
```



Tilde Expansion

- The tilde character ('∼') has a special meaning
- Used in a path, it expands into the name of the home directory
- ∼ is replaced by /home/duck (if your user name is *duck*)

```
$ pwd
/home/duck/myMusic/adele
$ cd ~/documents
$ pwd
/home/duck/documents
```

cd – bonus material

- `cd` without arguments changes into your home directory:

```
$ pwd
/home/duck/myMusic/adele
$ cd
$ pwd
/home/dock
```

- `cd -` changes into the *previous* working directory

```
$ cd -
$ pwd
/home/duck/myMusic/adele
```

Command: `ls` – list the content of directories

- `ls` lists contents of a directory
- Without an argument, shows the contents of the working directory

```
$ pwd
/home/duck/myMusic
$ ls
adele  coldplay.mp3   song.mp3
```

Command: `ls` – list the content of directories

- `ls` lists contents of a directory
- Without an argument, shows the contents of the working directory

```
$ pwd
/home/duck/myMusic
$ ls
adele  coldplay.mp3   song.mp3
```

- Many options available, e. g. :
  - -**l**, uses a long listing format (permissions, owner, size, mod. times)
  - -**a**, 'show all', forces ls to show all files and directories
  - -**h**, displays the file sizes in human readable format
  - -**F**, adds a trailing '/' to the names of directories
  - -**r**, reverses the order while sorting

ls – examples

```
$ pwd
/home/duck/myMuic
$ ls -a  # Shows hidden files
.  ..  adele  coldplay.mp3  song.mp3
$ ls -l  # "long" format
total 8
drwxrwxr-x 2 duck duck   6 Oct 10 11:34 adele
-rw-rw-r-- 1 duck duck  29 Oct 10 11:34 coldplay.mp3
-rw-rw-r-- 1 duck duck 119 Oct 10 11:34 song.mp3
$ ls -hl  # human-readable sizes
total 8.0K
drwxrwxr-x 2 duck duck   6K Oct 10 11:34 adele
-rw-rw-r-- 1 duck duck  29K Oct 10 11:36 coldplay.mp3
-rw-rw-r-- 1 duck duck 119K Oct 10 11:37 song.mp3
```

Command: `mkdir` – make directory

- `mkdir` (make directory) creates new (empty) directories
- Usage: mkdir [OPTION] NAME

```
$ pwd
/home/duck
$ ls -F
data/  myMusic/
$ mkdir myPublications
$ ls -F
data/  myMusic/  myPublications/
```

- If the directory already exists, it reports an error

```
$ mkdir data
mkdir: cannot create directory 'data': File exists
```

`mkdir -p`

- `mkdir -p` – no error if existing, make parent directory if needed

```
$ ls thesis
ls: cannot access 'thesis': No such file or directory

$ mkdir thesis/references
mkdir: cannot create directory 'thesis/references': No
such file or directory

$ mkdir -p thesis/references
$ ls -F
thesis/
$ ls -F thesis
references/
```

Command: `rmdir` – remove directory

- `rmdir` removes an empty directory
- Usage: `rmdir <directory name>`

```
$ cd ~/myPublications/thesis
$ ls -F
references/
$ rmdir references
$ ls
$
```

Interlude: success and failure

Rule of thumb

- If a command fails, you get an error message:
    - File not found
    - Disk full
    - ...
- On success, *no message is printed*

There are many exceptions, but
no output usually means that everything is fine

Exercise

- Create a folder
- Change into it
- Delete it (watch out: you cannot delete a folder while you are in it)
- Optional: Try to create (and delete) multiple nested subfolders
- Use `pwd` and `ls` as necessary

Exercise

- Create a folder
- Change into it
- Delete it (watch out: you cannot delete a folder while you are in it)
- Optional: Try to create (and delete) multiple nested subfolders
- Use `pwd` and `ls` as necessary

```
$ mkdir myfolder
$ cd myfolder
$ cd ..
$ rmdir myfolder
$ mkdir -p folder2/nestedfolder
# or:
$ mkdir folder2
$ cd folder2
$ mkdir nestedfolder
```

Hidden files

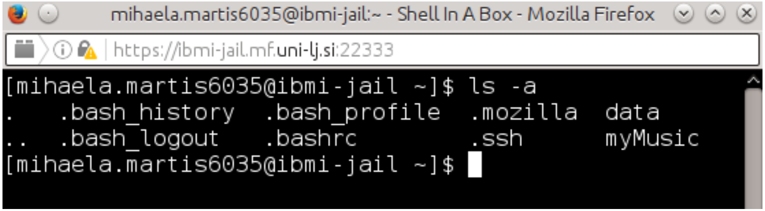
- If a file name starts with a dot, the file is *hidden*
- Configuration files are often hidden. Example: `/home/user/.bashrc` is a Bash configuration file
- A normal `ls` does not show hidden files
- Use **`ls -a`** to display them

Hidden files

- If a file name starts with a dot, the file is *hidden*
- Configuration files are often hidden. Example: `/home/user/.bashrc` is a Bash configuration file
- A normal `ls` does not show hidden files
- Use **`ls -a`** to display them
- The `.` and `..` directories are hidden

Hidden files

- If a file name starts with a dot, the file is *hidden*
- Configuration files are often hidden. Example: `/home/user/.bashrc` is a Bash configuration file
- A normal `ls` does not show hidden files
- Use **`ls -a`** to display them
- The `.` and `..` directories are hidden


mihaela.martis6035@ibmi-jail:~ - Shell In A Box - Mozilla Firefox
https://ibmi-jail.mf.uni-lj.si:22333

```
[mihaela.martis6035@ibmi-jail ~]$ ls -a
.    .bash_history  .bash_profile  .mozilla  data
..   .bash_logout   .bashrc        .ssh      myMusic
[mihaela.martis6035@ibmi-jail ~]$ 
```

Command: `tree`

- `tree` – list contents of directories in a tree-like format

```
$ pwd
/home/duck/myMusic
$ tree
.
|-- adele
|-- coldplay.mp3
|-- song.mp3
|-- song1.mp3
|-- song2.mp3
|-- song3.mp3
`-- song4.mp3
1 directory, 6 files
```

Command: `tree`

- `tree` – list contents of directories in a tree-like format

```
$ pwd
/home/duck/myMusic
$ tree
.
|-- adele
|-- coldplay.mp3
|-- song.mp3
|-- song1.mp3
|-- song2.mp3
|-- song3.mp3
`-- song4.mp3
1 directory, 6 files
```

- `tree -d` – list directories only

```
$ tree -d
.
`-- adele
1 directory.
```

Command: cp – copying files

- `cp` copies one or more files
- `cp source... destination`

```
$ cp coldplay.mp3 coldplay2.mp3
$ ls
coldplay.mp3 coldplay2.mp3

$ cp coldplay.mp3 ../coldplay_backup.mp3
$ ls ..
coldplay.mp3  data  myMusic  myPublications
```

- `cp -r` copies *recursively*

```
$ cp -r adele_songs /home/duck/backup
$ ls /home/duck/backup
adele_songs
```

Command: mv – move and rename files

- `mv` moves files or directories from one location to another
- Usage: `mv <file-or-directory> <destination>`

```
$ ls
adele  coldplay.mp3  song.mp3
$ mv coldplay.mp3 /home/cat/myProject
$ ls
adele  song.mp3
$ ls /home/cat/myProject
coldplay.mp3
```

Command: mv – move and rename files

- `mv` moves files or directories from one location to another
- Usage: `mv <file-or-directory> <destination>`

```
$ ls
adele  coldplay.mp3  song.mp3
$ mv coldplay.mp3 /home/cat/myProject
$ ls
adele  song.mp3
$ ls /home/cat/myProject
coldplay.mp3
```

- `mv` renames a file or a directory

```
$ ls
adele  song.mp3
$ mv adele adele_songs
$ ls
adele_songs  song.mp3
```

Removing files and directories

- `rm` removes files (not directories)

```
$ rm draft.txt
```

- Use `-i` to ask for confirmation

```
$ rm -i draft.txt
rm: remove regular file 'draft.txt'? yes
```

- `-r` removes directories *recursively*. That is, the directory and everything in it

```
$ rm thesis/references
rm: cannot remove 'thesis/references/': Is a directory
$ rm -r thesis/references
```

- ⚠ The shell **does not** have a trash bin → deleted files cannot be recovered

Removing files and directories

- `rm` removes files (not directories)

```
$ rm draft.txt
```

- Use `-i` to ask for confirmation

```
$ rm -i draft.txt
rm: remove regular file 'draft.txt'? yes
```

- `-r` removes directories *recursively*. That is, the directory and everything in it

```
$ rm thesis/references
rm: cannot remove 'thesis/references/': Is a directory
$ rm -r thesis/references
```

- ⚠ The shell **does not** have a trash bin → deleted files cannot be recovered
- If you expect a directory to be empty, use `rmdir` instead of `rm -r`

Wildcards

- Many commands can work with many files at once (for example, `ls`)
- But it would be tedious to type out all the file names
- *Wildcards* make this easier. They allow you to type only the common part of all names

```
$ ls *.mp3
coldplay.mp3  song.mp3
```

- The `*` is a wildcard that matches zero or more characters
- Before running the `ls` command, the `*.mp3` argument is replaced by all file names that match `*.mp3`

Basic wildcards

- * – matches zero or more characters
- ? – matches exactly one character
- [ ] – matches a range of characters (0-9, a-Z)
- { } – specify a list of terms separated by commas
- \ – used as an ‘escape’ character

Wildcard examples I

- List all files ending with '.mp3' or starting with 'c'

```
$ ls *.mp3
coldplay.mp3  song.mp3

$ ls c*
coldplay.mp3
```

- List all files named 'song', followed by a single character and ending with '.mp3'

```
$ ls
adele  coldplay.mp3  song.mp3  song1.mp3  song2.mp3
    song3.mp3  song4.mp3

$ ls song?.mp3
song1.mp3  song2.mp3  song3.mp3  song4.mp3
```

Wildcard examples II

- List all files with the numbers 1 or 2 in their names

```
$ ls *[1-2]*
song1.mp3  song2.mp3
```

- List all files starting with a 'c' and those containing the numbers 2 to 4 in their name

```
$ ls {c*,*[2-4]}.mp3
coldplay.mp3  song2.mp3  song3.mp3  song4.mp3
```

Exercise

1. Create a folder `exercise2`
2. Copy all `.txt` files from `~/data/others/` into it
3. Change into the directory you created
4. Remove all files in `exercise2` that have `employee` in their name
5. Rename `animals.txt` to `humans.txt`

Exercise

1. Create a folder `exercise2`
2. Copy all `.txt` files from `~/data/others/` into it
3. Change into the directory you created
4. Remove all files in `exercise2` that have `employee` in their name
5. Rename `animals.txt` to `humans.txt`

```
$ mkdir exercise2
$ cp ~/data/others/*.txt exercise2
$ cd exercise2
$ rm *employee*
$ mv animals.txt humans.txt
```

Summary

- The **filesystem** is hierarchical
- An **absolute path** starts from the root (`/`)
- A **relative path** starts relative to another directory, usually the **working directory**
- **Wildcards** save typing

Summary

- The **filesystem** is hierarchical
- An **absolute path** starts from the root (/)
- A **relative path** starts relative to another directory, usually the **working directory**
- **Wildcards** save typing
- Commands we learned:
  - **pwd** – display path to the working directory
  - **cd** – change directory
  - **ls** – list the content of directories
  - **mkdir** – create new directories
  - **rm** – remove files and directories
  - **rmdir** – remove empty directories
  - **cp** – copy files and directories
  - **mv** – move and/or rename a file or directory
  - **tree** – list contents of directories in a tree-like format.