Files and Directories

Working with directories

Print working directory (pwd)

The first command we will look at is the `pwd` command. The manpages gives an accurate description of what the command does:

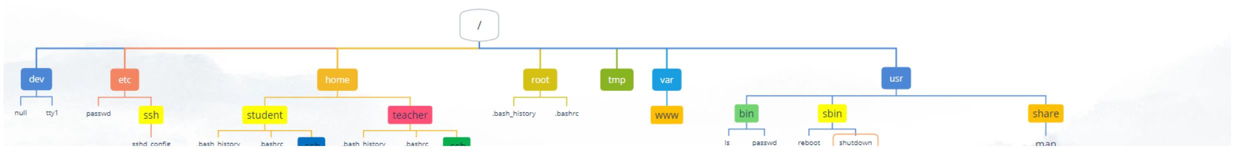
```
student@linux-ess:~$ whatis pwd
pwd - print name of current/working directory
```

This command prints the current working directory. The working directory refers to the directory where the prompt is active in.

```bash
student@linux-ess:~$ pwd
/home/student
```

In Windows, an *absolute path* starts with `C:\...` . In Linux we do not use Drive letters. The C:\ drive in Windows is called the *root directory* in Linux. This directory is reffered to as a `/` at the beginning of a path. More about *absolute* and *relative* paths later in this chapter.

The folder `student` is our current working directory. This folder is a subfolder of the folder `home` which on his turn is a subfolder of the *root directory* `/` .



Remember the prompt which contained an active path? The `~` sign was an abbreviation for the folder `/home/student`. This is called the *homefolder*. Every user on the operating system will get their own *homefolder* in the folder `/home` just like in Windows, where every user has his own folder under `C:\Users`. A user has all permissions (read, write, execute) in his own homefolder. Outside of that folder he often only has *read* permissions.

Change working directory (cd)

You can change the current working directory with the `cd` command (change directory):

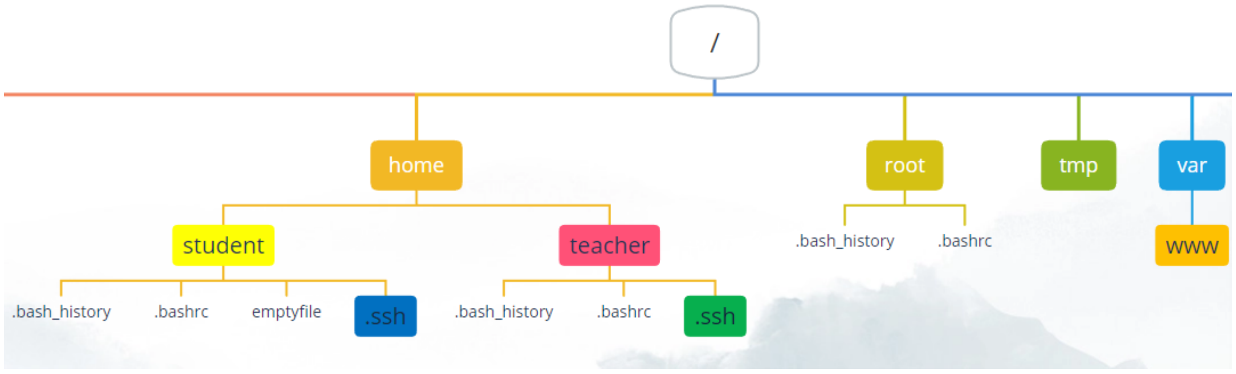
```bash
student@linux-ess:~$ pwd
/home/student
student@linux-ess:~$ cd /etc
student@linux-ess:/etc$ pwd
/etc
student@linux-ess:/etc$ cd ~
student@linux-ess:~$ pwd
/home/student
```

The command takes a path as an argument. In the example above we will navigate to the folder `etc` on the root directory `/`. We can see this in the output of the `pwd` command and in the prompt definition. As shown in the second part of the example above we can also use the `~` sign to quickly navigate to the user's homefolder.

We could also use the `cd` command without any argument. This will navigate back to the users homefolder as well:

bash

```
student@linux-ess:~$ cd /home/student
student@linux-ess:~$ pwd
/home/student
student@linux-ess:~$ touch emptyfile
student@linux-ess:~$ ls
emptyfile
student@linux-ess:~$ cd /etc
student@linux-ess:/etc$ pwd
/etc
student@linux-ess:/etc$ cd
student@linux-ess:~$ pwd
/home/student
student@linux-ess:~$ ls
emptyfile
```



The command `touch` is used to create an empty file named `emptyfile`. The command is explained later on in this chapter.

Display a tree view in the shell (tree)

You can display a tree view of a directory with its subdirectories with the `tree` command:

bash

```
student@linux-ess:~$ tree
.
```

```
└── emptyfile

0 directories, 1 file
```

ℹ If you are prompted that tree isn't yet installed just run the command he is proposing.

We could also use the `-a` option to view the hidden files (files starting with a dot):

```bash
student@linux-ess:~$ tree -a
.
├── .bash_history
├── .bash_logout
├── .bashrc
├── .cache
│   └── motd.legal-displayed
├── .lesshst
├── .local
│   └── share
│       └── nano
├── .profile
├── emptyfile
├── .ssh
│   └── authorized_keys
├── .sudo_as_admin_successful
└── .wget-hsts

5 directories, 10 files
```

We can also specify a path as a parameter to get a view of a certain directory:

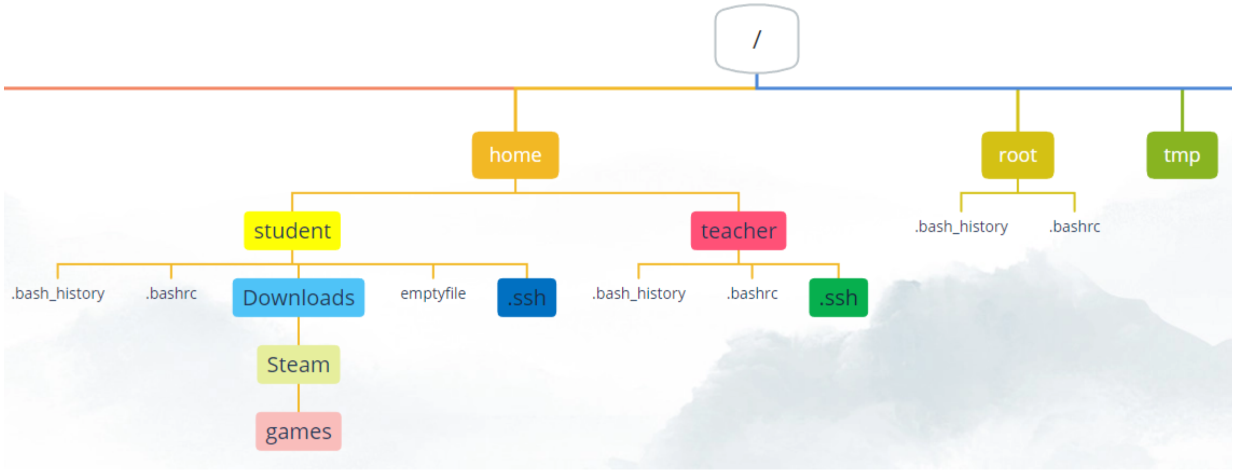
```bash
student@linux-ess:~$ tree /etc/dhcp
/etc/dhcp
```

```
├── debug
├── dhclient.conf
├── dhclient-enter-hooks.d
│   └── debug -> ../debug
└── dhclient-exit-hooks.d
    ├── debug -> ../debug
    ├── hook-dhclient
    ├── resolved
    ├── rfc3442-classless-routes
    └── timesyncd

2 directories, 8 files
```

Absolute & relative paths

When using commands we often have to use paths to certain folders or files. The `cd` command for example requires a path to navigate to.

Paths are used to point towards files and folders on the filesystem. We can use two types of paths: *relative* and *absolute* paths.



Absolute path

Absolute paths must start with a `/` sign. This means that an absolute path will start from the *root* (`/`) directory (the highest level on the filesystem) and will work its way down. For example:

bash

```
student@linux-ess:/etc$ pwd
/etc
student@linux-ess:/etc$ cd /home/student/
student@linux-ess:~/$ pwd
/home/student
```

In Windows an absolute path will start with `C:\` rather than `/` !

Did you know the Linux CLI has command and pathcompletion? Try pressing the `tab` key when typing part of a command, file or folder name. If any command, file or folder matches the first part of the text you typed it will automatically complete the word for you!

When using tab competion on a directory it will end with a slash (/), e.g. `cd /home/student/` . The last slash (/) is optional, so it is exactly the same as `cd /home/student`

Relative path

A relative path will always start from *the current working directory* and will point to another file or folder from there. For example:

bash

```
student@linux-ess:~$ pwd
/home/student
student@linux-ess:~$ cd Downloads/Steam/games
student@linux-ess:~/Downloads/Steam/games$ pwd
/home/student/Downloads/Steam/games
```

When using relative paths, we can use some shortcuts:

```
.(one dot): Refers to the current directory
..(two dots): Refers to the parent directory
~ (tilde): Refers to the current user's homefolder
```

This means that, when in the folder `/home/student/Downloads` , we could use `..` to navigate to the parent directory `/home/student`

```bash
student@linux-ess:~/Downloads$ pwd
/home/student/Downloads
student@linux-ess:~/Downloads$ cd ..
student@linux-ess:~$ pwd
/home/student
```

We could integrate these shortcuts in relative paths as well:

```bash
student@linux-ess:/etc$ pwd
/etc
student@linux-ess:/etc$ cd /home/student/Downloads # Absolute path
student@linux-ess:~/Downloads$ pwd
/home/student/Downloads
student@linux-ess:~/Downloads$ cd ../../teacher # Relative path
student@linux-ess:/home/teacher$ pwd
/home/teacher
```

ℹ A number sign `#` tells the shell that everything behind it is considered a comment and will not be interpreted as a command or argument!

Listing directory contents (ls)

To list the contents of a directory, we can use the `ls` command. Using the command without any options or arguments will list de contents of the current working directory:

bash

```
student@linux-ess:~$ ls
emptyfile  Downloads
```

We can also go in a certain directory to list its contents:

bash

```
student@linux-ess:~$ cd /
student@linux-ess:/$ ls
bin   dev  home  lib32  libx32      media  opt   root  sbin  srv  tmp  va
boot  etc  lib   lib64  lost+found  mnt    proc  run   snap  sys  usr
```

The `ls` command can also take one argument. This argument is a path which can be absolute or relative. The `ls` command will then show the contents of that folder.

bash

```
student@linux-essentials:~$ ls
emptyfile  Downloads
student@linux-essentials:~$ ls /
bin   dev  home  lib32  libx32      media  opt   root  sbin  srv  tmp  va
boot  etc  lib   lib64  lost+found  mnt    proc  run   snap  sys  usr
student@linux-essentials:~$
```

The `ls` command has different options as well. The options can be found in the manpage using `man ls`. For example:

bash

```
student@linux-ess:~$ ls -alh .       # The dot sign refers to the current
total 45M
drwxr-xr-x 5 student student 4.0K Mar 27 16:36 .
drwxr-xr-x 3 root    root    4.0K Oct  5 13:40 ..
-rw------- 1 student student 1.7K Mar 17 12:14 .bash_history
-rw-r--r-- 1 student student  220 Oct  5 13:40 .bash_logout
```

```
-rw-r--r-- 1 student student 3.7K Oct  5 13:40 .bashrc
drwxrwxr-x 2 student student 4.0K Oct  1 14:31 Downloads
-rw-r--r-- 1 student student    0 Feb 28 11:48 emptyfile
-rw-r--r-- 1 student student  807 Oct  5 13:40 .profile
drwxr-xr-x 2 student student 4.0K Oct  5 13:40 .ssh
-rw-r--r-- 1 student student    0 Oct  6 08:20 .sudo_as_admin_successful
```

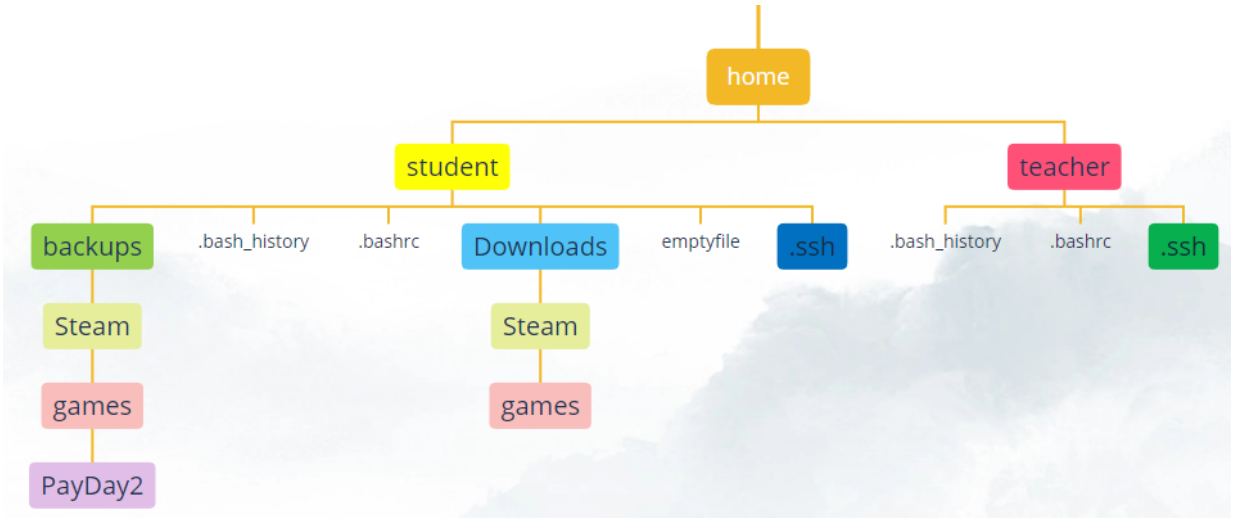
Notice how we combined 3 options in the command above. Both `ls -a -l -h` and `ls -alh` will function exactly the same and will use all 3 options. The options can be put in any order, so `ls -hal` is also correct. These 3 options are used most often when it comes to the `ls` command. You could search for them in the manpage but we will give an overview:

- the `-a` option will *also show hidden files and folders*. In Linux, hidden files and folders start with a dot (.). e.g. the `.bashrc` file.
- the `-l` option will show a *long listing*. This means that it will show all the extra output* and not just the file and folder names.
- the `-h` option refers to *human readable* and will make filesizes appear with the proper measuring unit rather than showing all sizes in bytes.

* The `-rw-r--r--` column refers to permissions on that specific file/folder. We will explain this in the chapter `users & permissions`. The columns containing `student student` refer to the owner and groupowner of that specific file/folder and are linked to the permissions column. The `3.7K` on the line of the file .bashrc refers to the file size and `Oct 5 13:40` refers to the timestamp of the last modification of the file.

Everything in Linux is a file. Not just the files, but folders too! They are just defined as *special* files. Your hard disk? A file. A USB drive? A File. Hardware such as your keyboard? You guessed it, a file!

Create directories (mkdir)



To create new directories we can use the `mkdir` (make directory) command. The command takes a path as an argument:

```bash
student@linux-ess:~$ mkdir backups
student@linux-ess:~$ ls
backups  Downloads  emptyfile
```

In the example above the `mkdir` command will create a folder named `backups` in the current working directory (`~` or `/home/student`). The folder name here is a *relative path*.

paths with subdirectories

When using *relative* or *absolute* paths we could do the following:

```bash
student@linux-ess:~$ mkdir backups/Steam/games/PayDay2
mkdir: cannot create directory ‘backups/Steam/games/PayDay2’: No such fil
```

The command tries to make a folder named `PayDay2` in the folder `games` which is located in the folder `Steam` which is located in the folder `Backups` . However we get an error because the folders `games` or `Steam` do not exist. We can tell to create any missing subfolders in the path by using the `-p` option:

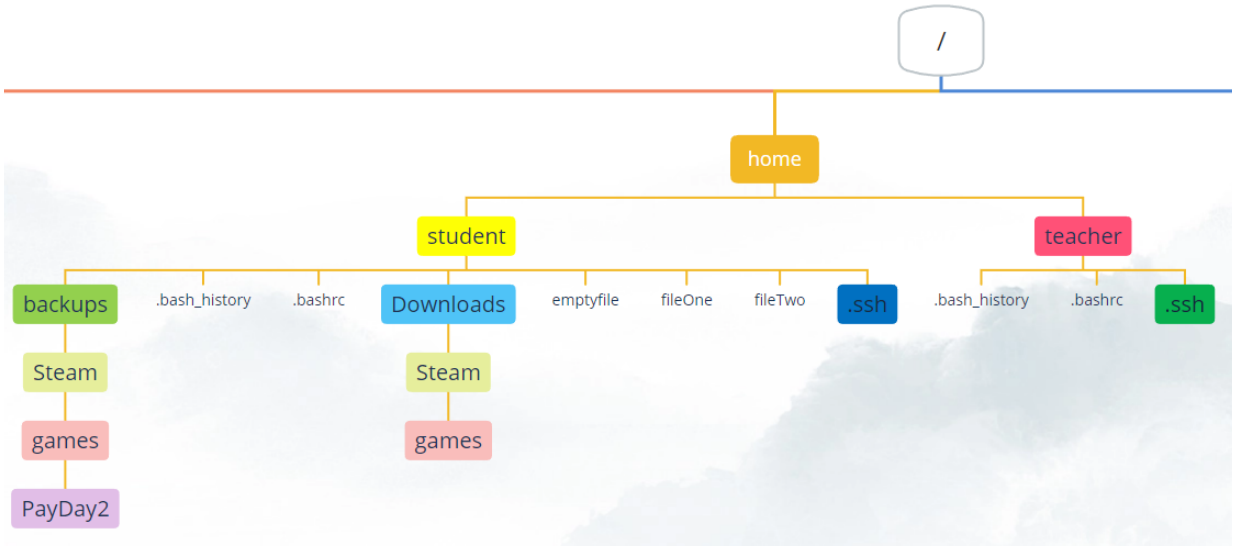
```
student@linux-ess:~$ mkdir -p backups/Steam/games/PayDay2
student@linux-ess:~$ tree backups
backups
└── Steam
    └── games
        └── PayDay2

3 directories, 0 files
```

This will create the folders `Steam` and `games` if they don't exist.

> ℹ As mentioned earlier everything in Linux is case sensitive. This is also applicable when creating files and folders. Try running the command `mkdir walkthroughs Walkthroughs` . This will create 2 folders: one with the name `walkthroughs` and one with the name `Walkthroughs` .

Working with files

Create an empty file (touch)

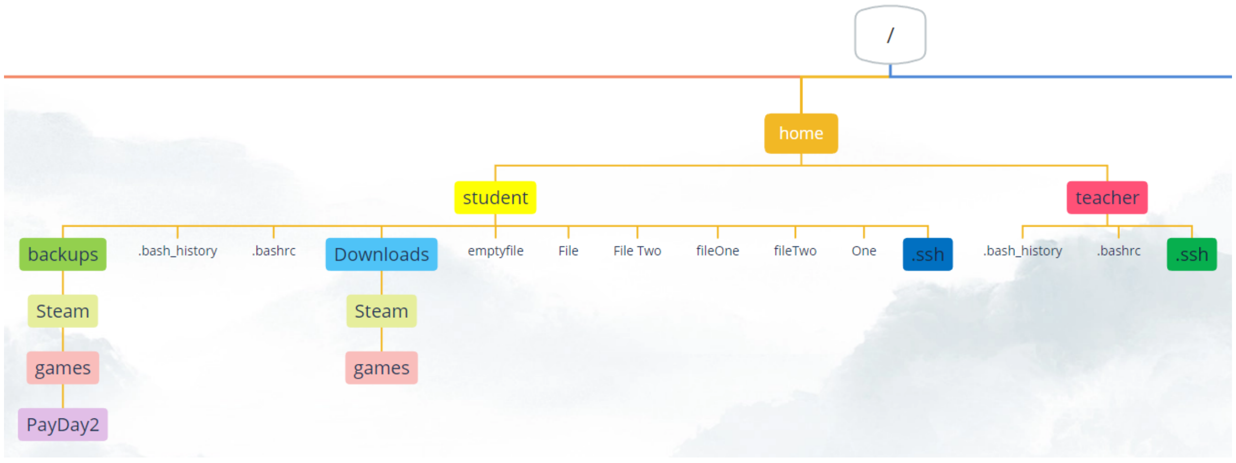


One easy way to create an empty file is with touch. The example starts with an empty directory, creates two files with touch and then lists those files:

```bash
student@linux-ess:~$ ls -l
total 8
drwxrwxr-x 3 student student 4096 Oct  1 14:34 backups
drwxrwxr-x 2 student student 4096 Oct  1 14:31 Downloads
-rw-rw-r-- 1 student student    0 Oct  1 14:31 emptyfile
student@linux-ess:~$ touch fileOne
student@linux-ess:~$ touch fileTwo
student@linux-ess:~$ ls -l
total 0
drwxrwxr-x 3 student student 4096 Oct  1 14:34 backups
drwxrwxr-x 2 student student 4096 Oct  1 14:31 Downloads
-rw-rw-r-- 1 student student    0 Oct  1 14:31 emptyfile
-rw-r--r-- 1 student student 0 Feb 12 09:50 fileOne
-rw-r--r-- 1 student student 0 Feb 12 09:50 fileTwo
```

Note that both of these files are empty as seen by the file size. In the next chapter we will look into ways to create files with contents.

Files with spaces in the name



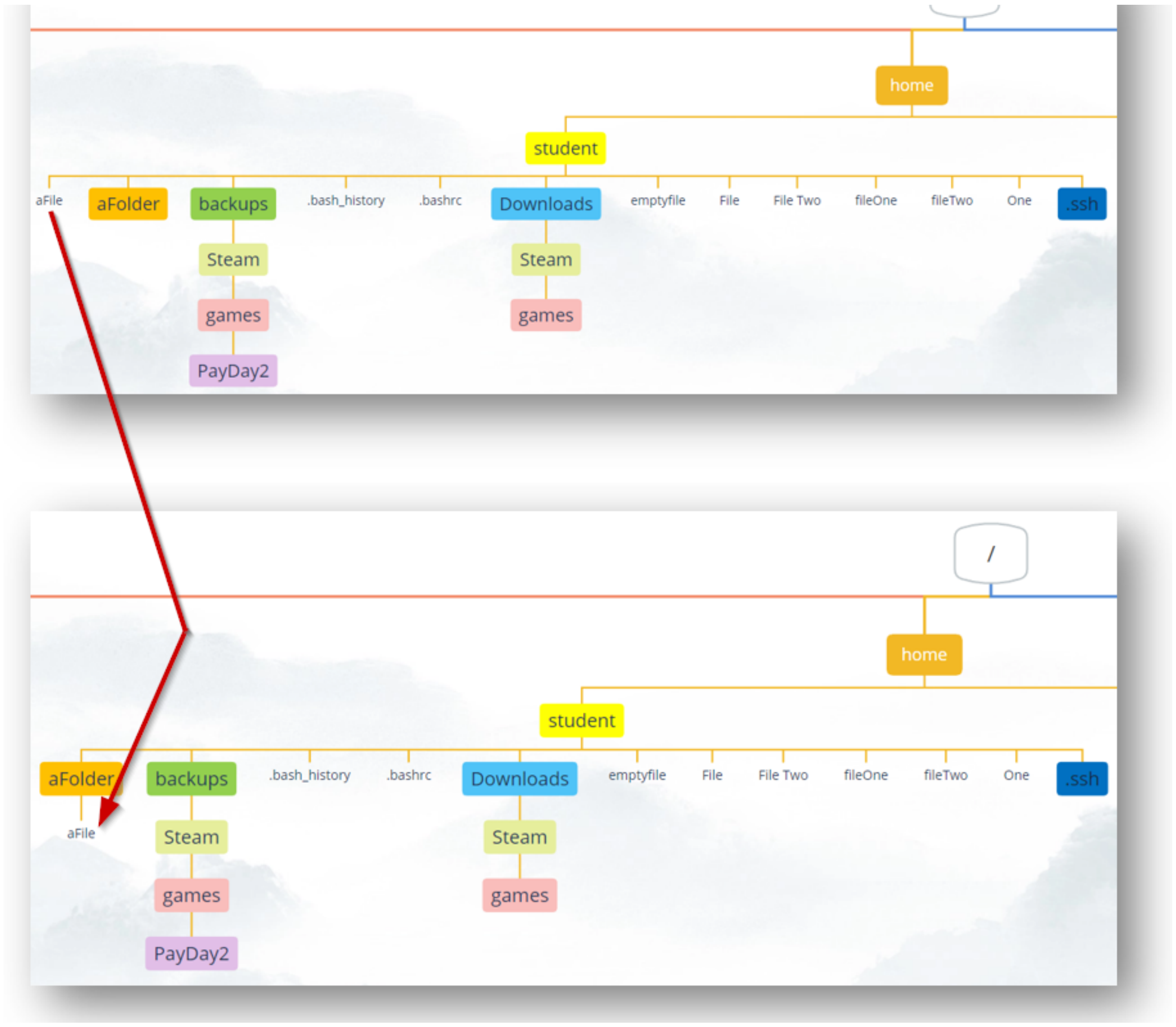
If we want to work with files with spaces in the name we can put the name between double quotes:

bash

```bash
student@linux-ess:~$ ls -l
total 0
drwxrwxr-x 3 student student 4096 Oct  1 14:34 backups
drwxrwxr-x 2 student student 4096 Oct  1 14:31 Downloads
-rw-rw-r-- 1 student student    0 Oct  1 14:31 emptyfile
-rw-r--r-- 1 student student 0 Feb 12 09:50 fileOne
-rw-r--r-- 1 student student 0 Feb 12 09:50 fileTwo
student@linux-ess:~$ touch File One
student@linux-ess:~$ ls -l
total 0
drwxrwxr-x 3 student student 4096 Oct  1 14:34 backups
drwxrwxr-x 2 student student 4096 Oct  1 14:31 Downloads
-rw-rw-r-- 1 student student    0 Oct  1 14:31 emptyfile
-rw-r--r-- 1 student student 0 Feb 12 09:50 fileOne
-rw-r--r-- 1 student student 0 Feb 12 09:50 fileTwo
-rw-r--r-- 1 student student 0 Feb 12 09:50 File
-rw-r--r-- 1 student student 0 Feb 12 09:50 One
student@linux-ess:~$ touch "File Two"
student@linux-ess:~$ ls -l
total 0
drwxrwxr-x 3 student student 4096 Oct  1 14:34 backups
drwxrwxr-x 2 student student 4096 Oct  1 14:31 Downloads
-rw-rw-r-- 1 student student    0 Oct  1 14:31 emptyfile
-rw-r--r-- 1 student student 0 Feb 12 09:50 fileOne
-rw-r--r-- 1 student student 0 Feb 12 09:50 fileTwo
-rw-r--r-- 1 student student 0 Feb 12 09:50 File
-rw-r--r-- 1 student student 0 Feb 12 09:50 'File Two'
-rw-r--r-- 1 student student 0 Feb 12 09:50 One
```

Note that we could also use single quotes `touch 'File Two'` or a backslash to escape the space `touch File\ Two` .

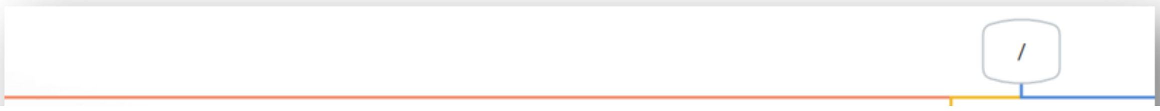
Move files (mv)

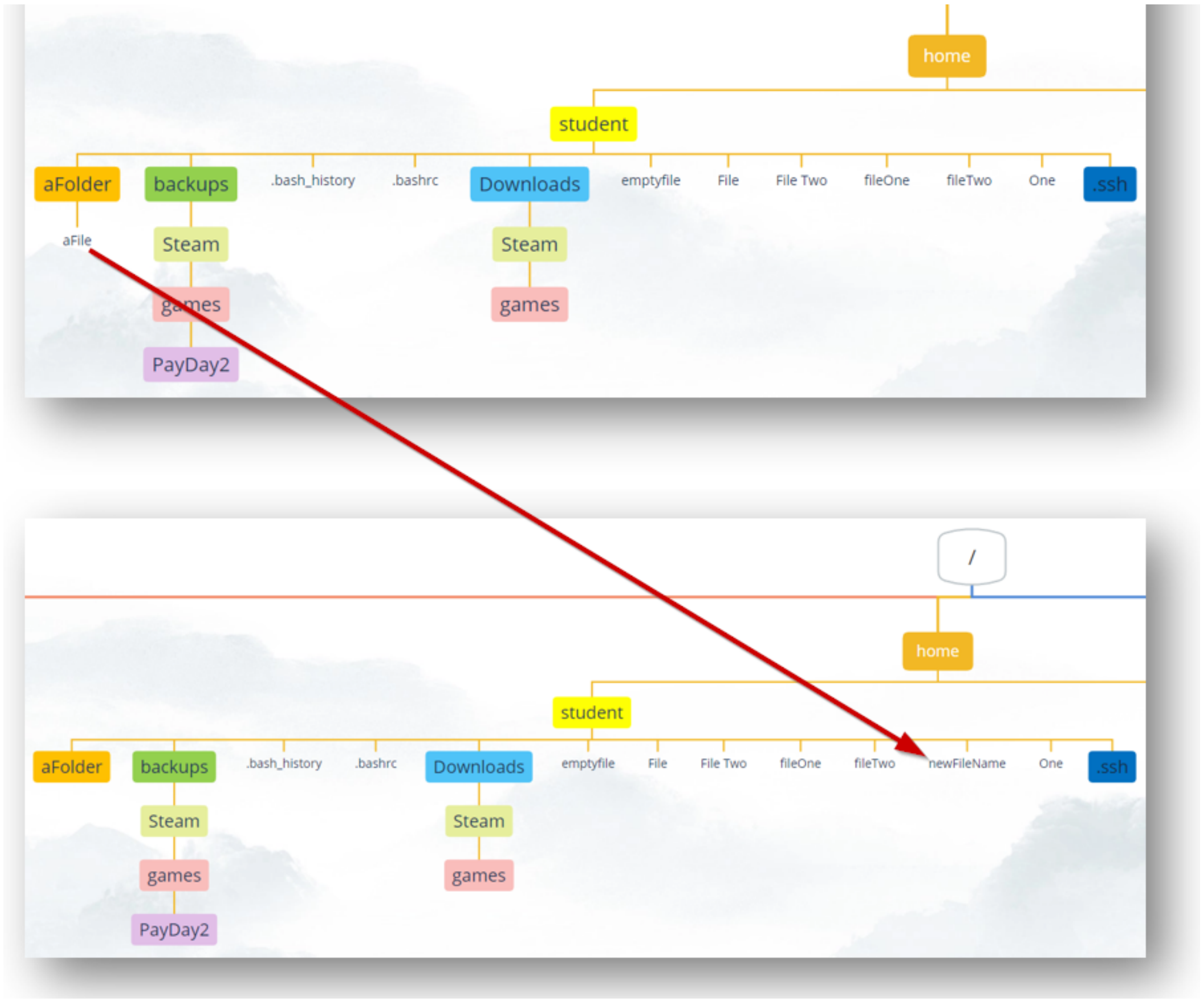


To move a file to another folder we can use the `mv` (move) command. This command takes two arguments: the source file/folder and the destination file/folder:

bash

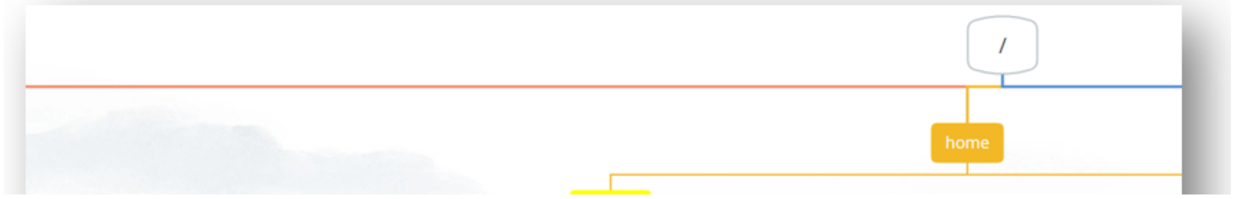
```
student@linux-ess:~$ ls
aFile  aFolder  backups  Downloads  emptyfile  File  fileOne  fileTwo  'F
student@linux-ess:~$ mv aFile aFolder/
student@linux-ess:~$ ls
aFolder  backups  Downloads  emptyfile  File  fileOne  fileTwo  'File Two
student@linux-ess:~$ ls aFolder
aFile
```

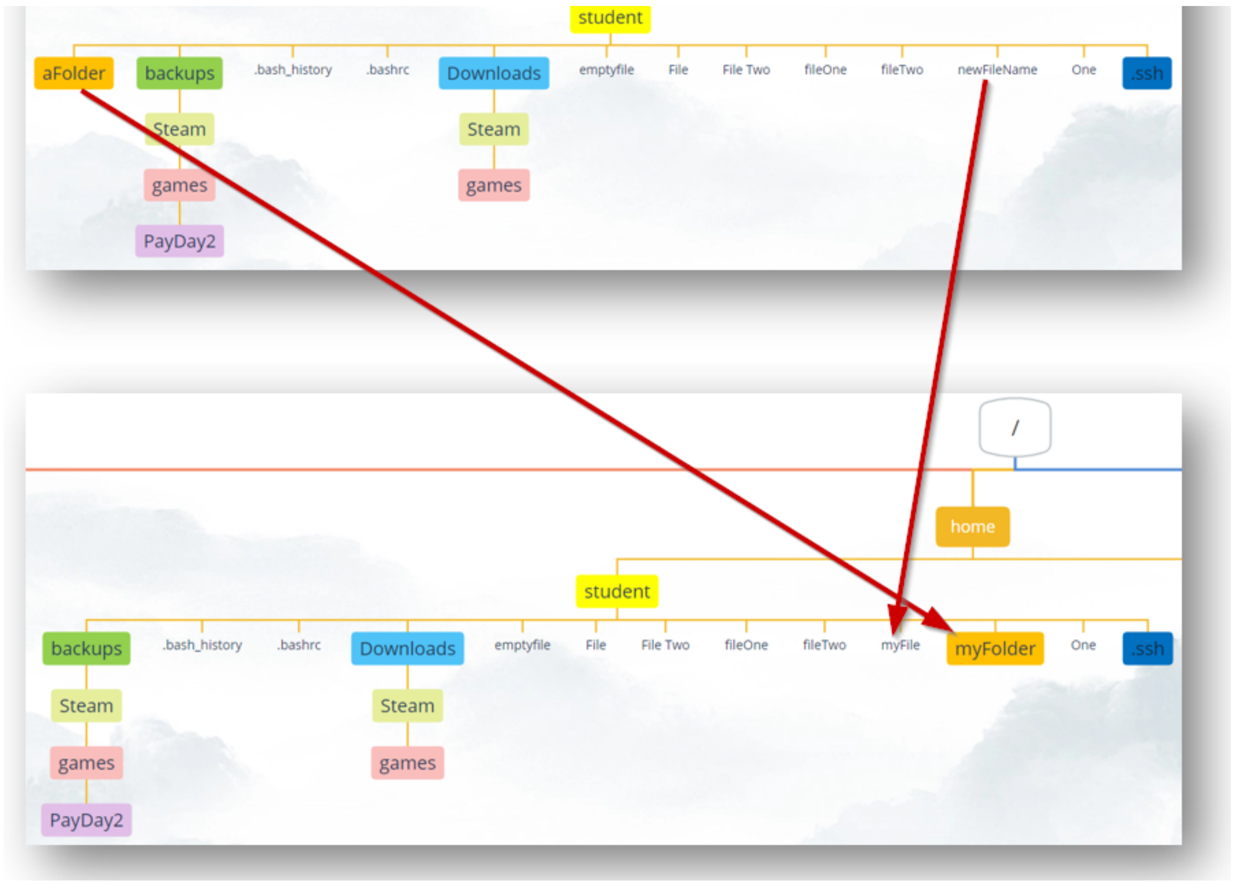




Since the second argument is a destination file or folder, we could use this command to rename a file as well. In the example below we use the `mv` command to move the file back to the homefolder but also renaming it:

```bash
student@linux-ess:~/aFolder$ ls
aFile
student@linux-ess:~/aFolder$ mv aFile ../newFileName
student@linux-ess:~/aFolder$ ls
student@linux-ess:~/aFolder$ ls ~
aFolder  backups  Downloads  emptyfile  File  fileOne  fileTwo  'File Two
```

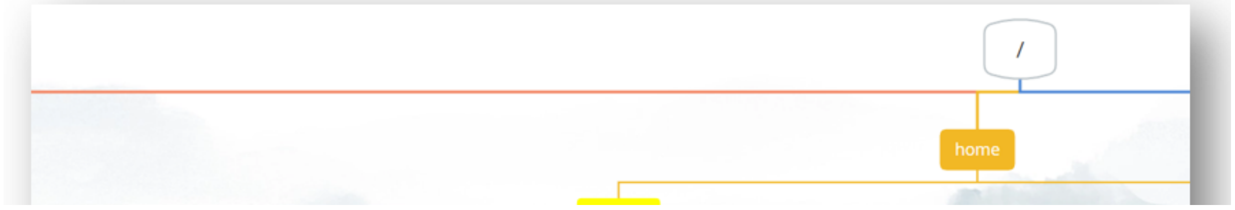


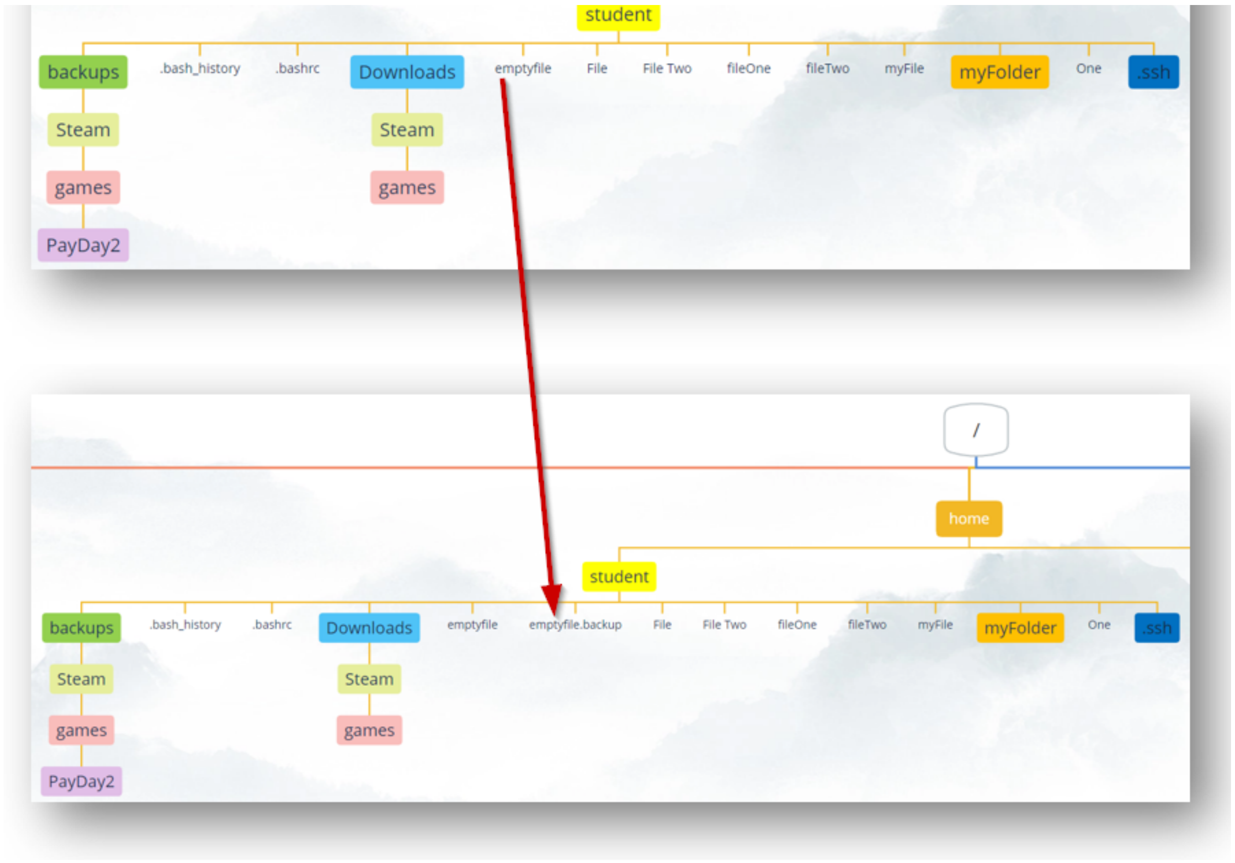


When renaming only one file or folder, `mv` is the prefered command to use. The examples both used files, but the same logic works for renaming and moving folders.

```bash
student@linux-ess:~$ ls
aFolder  backups  Downloads  emptyfile  File  fileOne  fileTwo  'File Two
student@linux-ess:~$ mv newFileName myFile
student@linux-ess:~$ mv aFolder myFolder
student@linux-ess:~$ ls
backups  Downloads  emptyfile  File  fileOne  fileTwo  'File Two'  myFile
```

Copy files (cp)



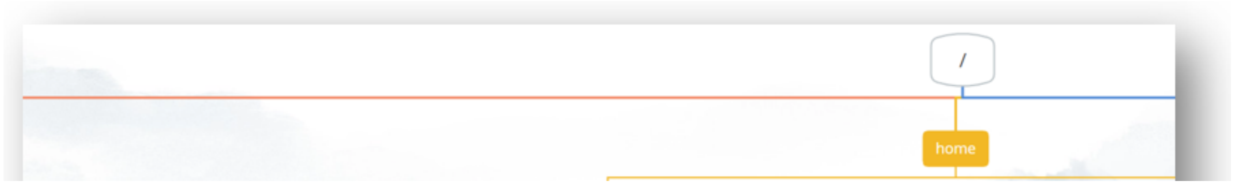


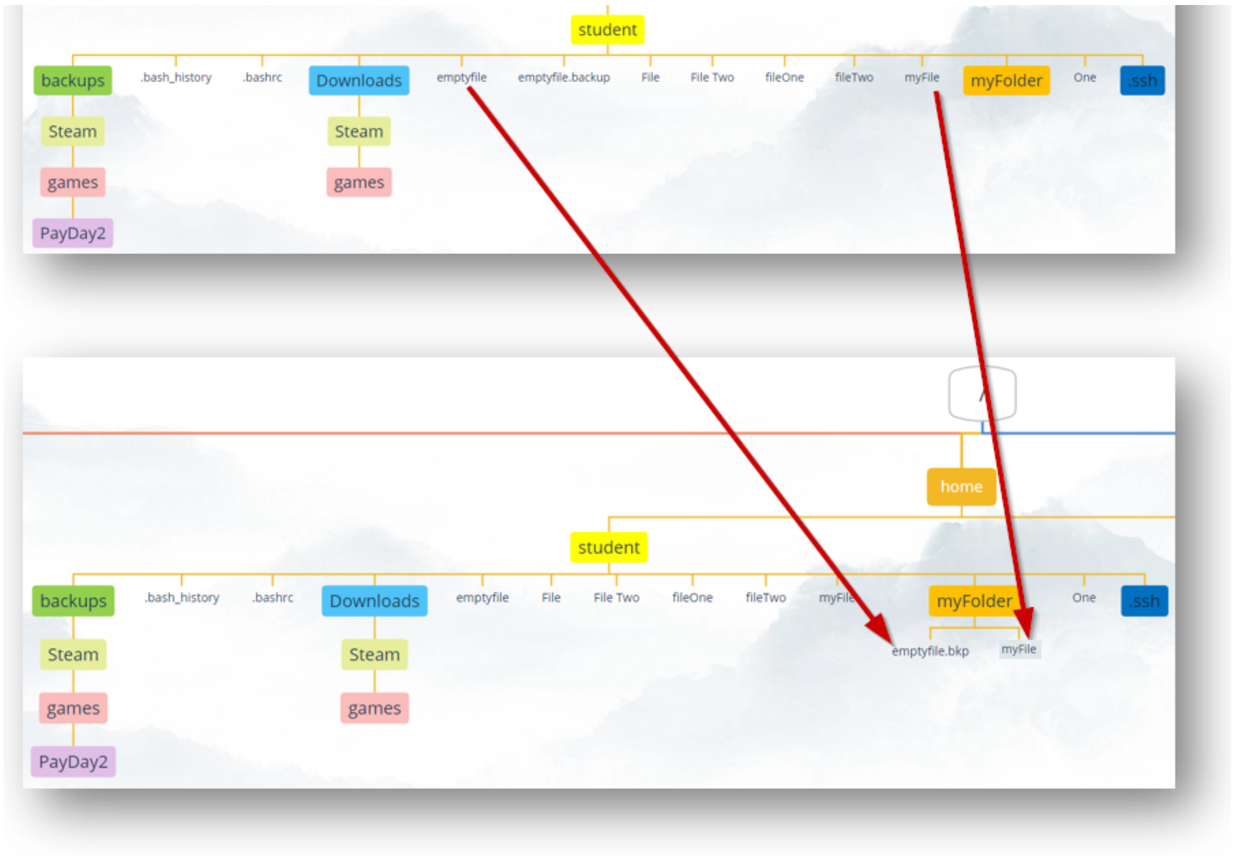
To make a copy of a file we can use the `cp` (copy) command as follows:

```bash
student@linux-ess:~$ ls
backups  Downloads  emptyfile  File  fileOne  fileTwo  'File Two'  myFile
student@linux-ess:~$ cp emptyfile emptyfile.backup
student@linux-ess:~$ ls
backups  Downloads  emptyfile  emptyfile.backup  File  fileOne  fileTwo
```

Both arguments are paths. The first path is the original file/folder. The second path is a path to the new location and (optional) filename/foldername. We can use the cp command to copy both files and folders to the same directory or a different directory.

Copy to another directory





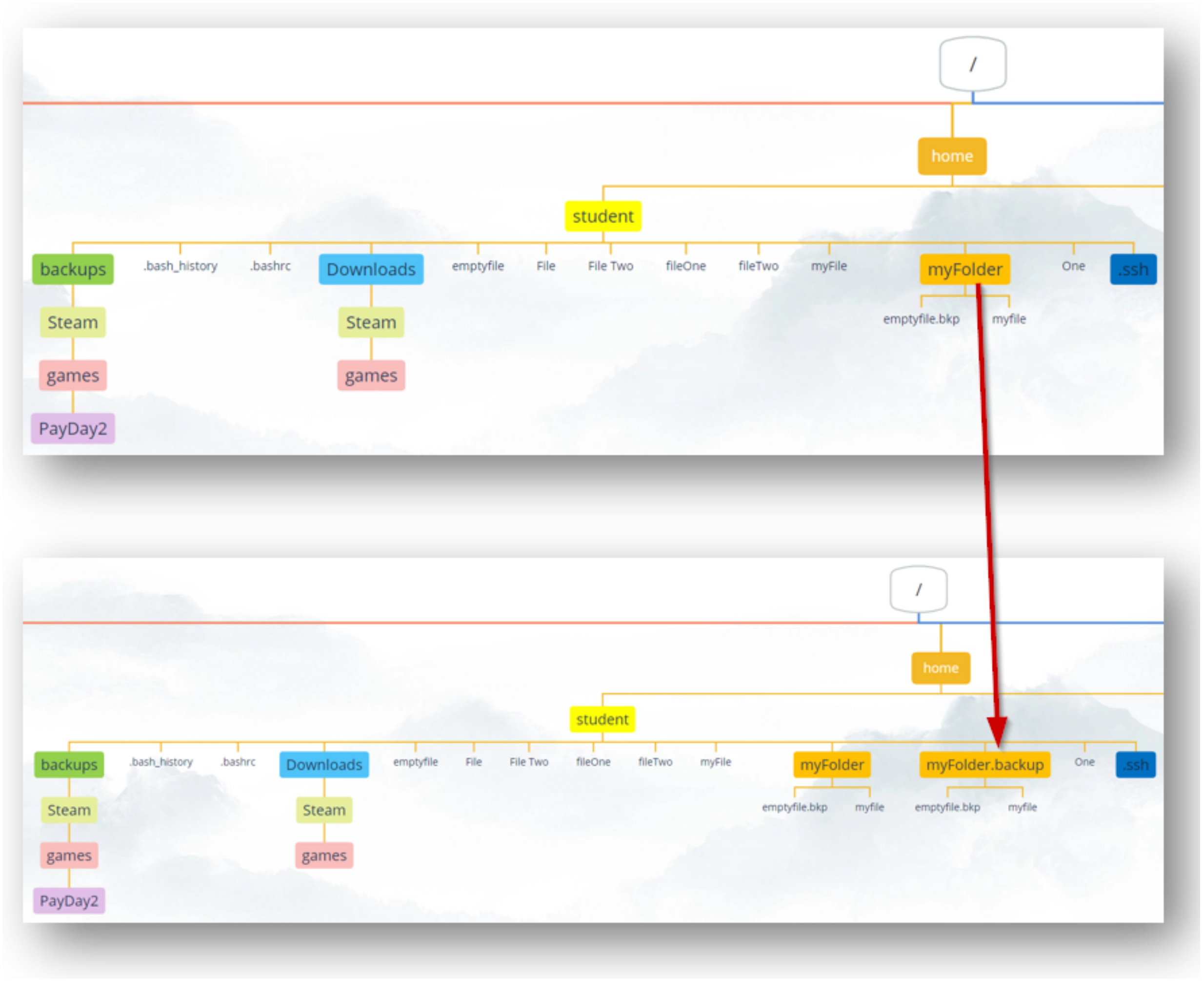
The example below shows us how we can use the arguments in the `cp` command to copy files or folders to another directory. In this example we copy the file named *myFile* into the directory called *myFolder* using a *relative* path. We also copy the file named *emptyfile* into the directory called *myFolder* and give it the new name *emptyFile.bkp*. We do this with using an absolute path.

bash

```
student@linux-ess:~$ ls
backups  Downloads  emptyfile  emptyfile.backup  File  fileOne  fileTwo
student@linux-ess:~$ cp myFile myFolder/
student@linux-ess:~$ ls myFolder
myFile
student@linux-ess:~$ ls
backups  Downloads  emptyfile  emptyfile.backup  File  fileOne  fileTwo
student@linux-ess:~$ cp /home/student/emptyFile /home/student/myFolder/em
student@linux-ess:~$ ls myFolder
myFile  emptyFile.bkp
student@linux-ess:~$ ls
backups  Downloads  emptyfile  emptyfile.backup  File  fileOne  fileTwo
```

ℹ Note that we can use both *relative* and *absolute* paths in the copy command for both the original file/folder and the destination file/folder.

Copy recursive



To copy complete directories (meaning all subfolders and files inside the directory) we will have to use the `-r` (recursive) option:

bash

```
student@linux-ess:~$ ls
backups  Downloads  emptyfile  emptyfile.backup  File  fileOne  fileTwo
student@linux-ess:~$ ls myFolder
myFile  emptyFile.bkp
student@linux-ess:~$ cp -r myFolder/ myFolder.backup
student@linux-ess:~$ ls
backups  Downloads  emptyfile  emptyfile.backup  File  fileOne  fileTwo
```

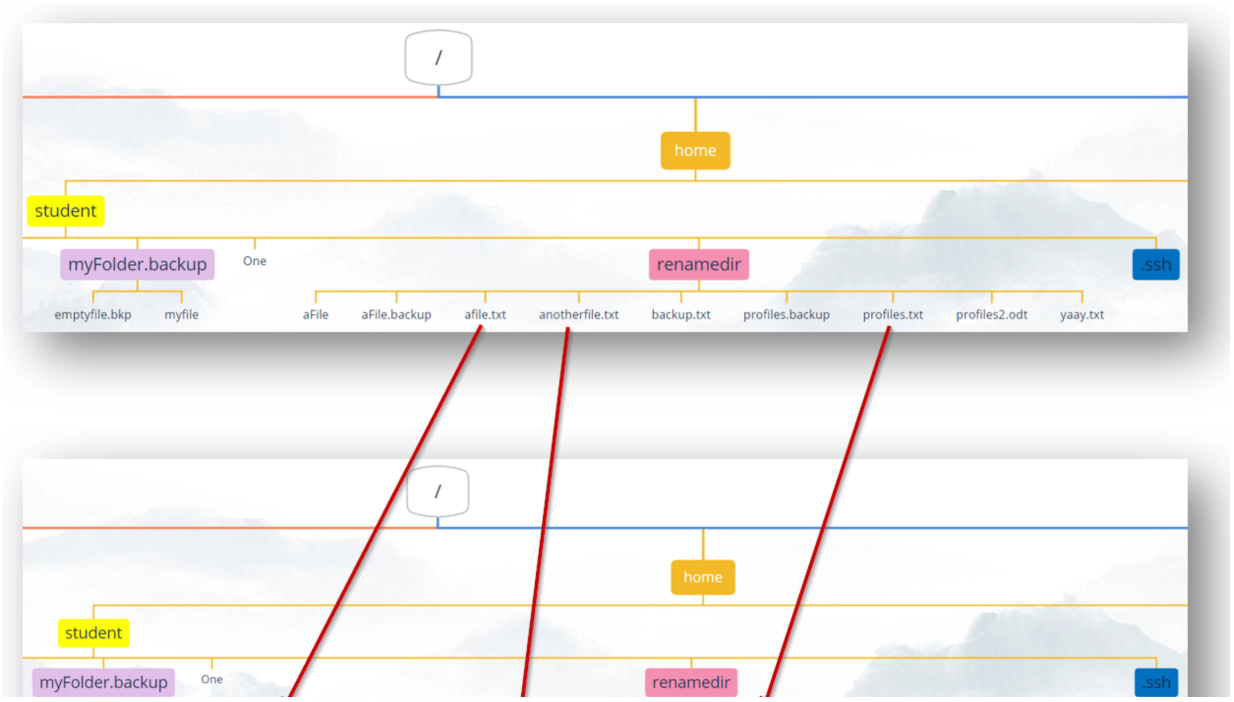
```
student@linux-ess:~$ ls myFolder.backup
myFile  emptyFile.bkp
```

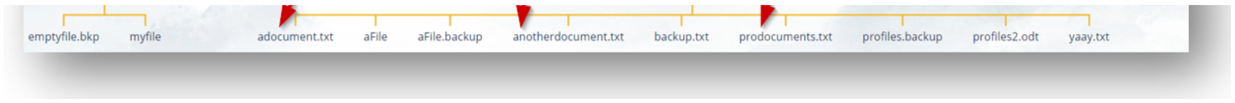
Overwrite files

We have to be aware that the `cp` command will overwrite existing files by default. We can use the `-i` (interactive) option to get a prompt where we have to confirm if we want to overwrite the file as seen in the example below:

```bash
student@linux-ess:~$ ls
backups  Downloads  emptyfile  emptyfile.backup  File  fileOne  fileTwo
student@linux-ess:~$ ls myFolder
myFile  emptyFile.bkp
student@linux-ess:~$ cp myFile myFolder/emptyFile.bkp     # No error, the
student@linux-ess:~$ ls myFolder
myFile  emptyFile.bkp
student@linux-ess:~$ cp -i myFile myFolder/emptyFile.bkp     # you will b
cp: overwrite `emptyFile.bkp'? y
```

Rename files (rename)





We saw that we could use the `mv` command to rename files and folders. This works and is often very easy but when you have to rename files in bulk you might want to consider another approach. The `rename` command is designed specifically to rename multiple files and folders with one command. To do this it uses a *regular expression* (Regex). A Regex is a sequence of characters that define a search pattern. We will learn more about regular expressions later in this course. It uses this search pattern to make certain changes to the filenames:

```bash
student@linux-ess:~$ mkdir renamedir
student@linux-ess:~$ cd renamedir
student@linux-ess:~/renamedir$ touch aFile  aFile.backup  afile.txt  anot
student@linux-ess:~/renamedir$ ls
aFile  aFile.backup  afile.txt  anotherfile.txt  backup.txt  profiles.bac
student@linux-ess:~/renamedir$ rename 's/file/document/' *.txt
student@linux-ess:~/renamedir$ ls
aFile  aFile.backup  adocument.txt  anotherdocument.txt  backup.txt  prod
```

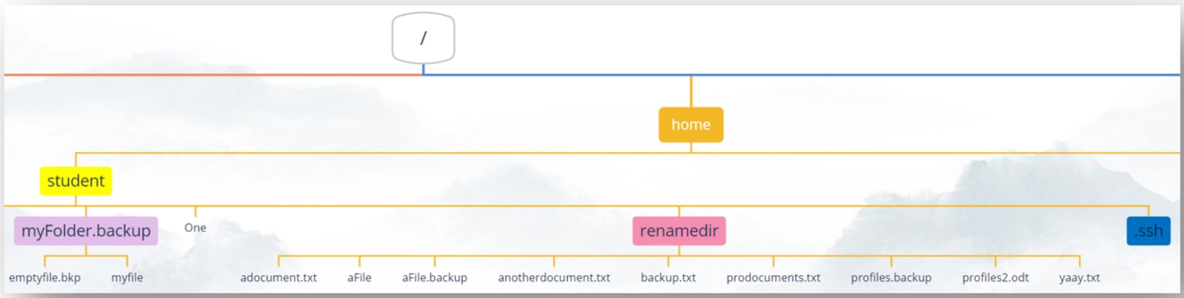
If rename is not yet installed you can do this with the command `sudo apt -y install rename` .

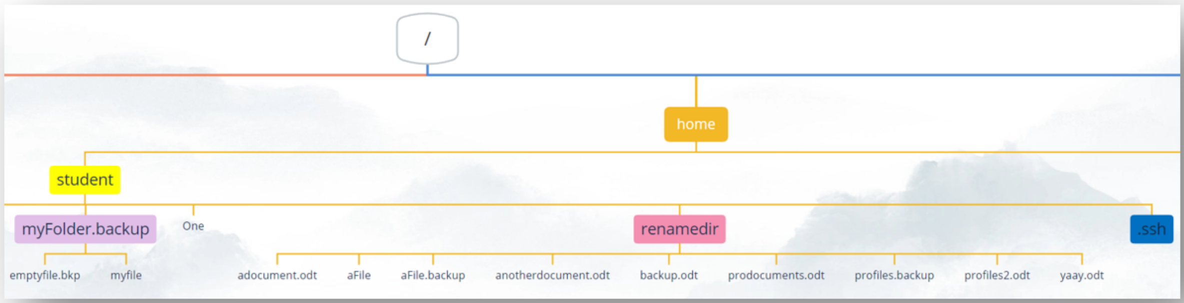
A lot is going on in the example above, let's summarize what is present:

- The folder we are in contains some `txt` files, some `backup` files and files with `no extension` . Some of the files contain the word `file` that we want to replace with `document` . Remember that by default Linux is case-sensitive, so `file` is not the same as `File` .
- the `rename` command takes a *string* with the value `s/file/document/` . This is the *regex* that is being used by the command to search (`s`)for names containing the word `file` and replace it with the word `document` .
- the last argument is `*.txt` . We use this to tell the `rename` command to only run the replacement regex on files ending in `.txt`

ℹ a `*` (asterisk) is considered a wildcard character in bash. It refers to *zero, one or more characters*. So in the example above this translates to: "run this replacement regex on all files containing zero,one or more characters of any kind, following and ending with the string `.txt` .

We could also use the `rename` command to change the file extentions of all files and folders:



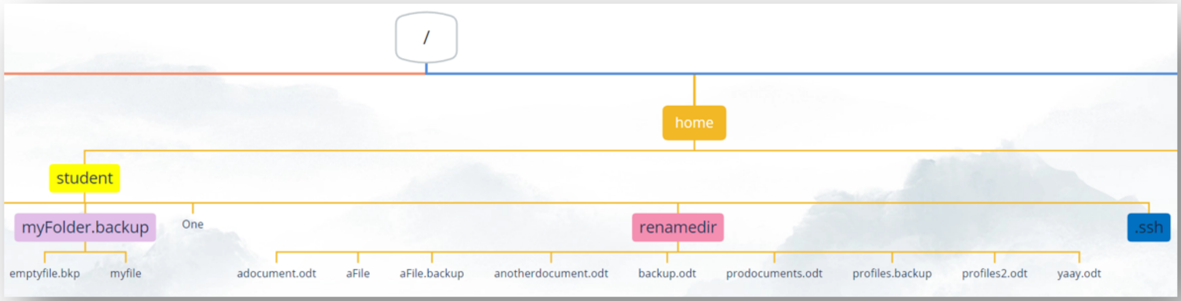


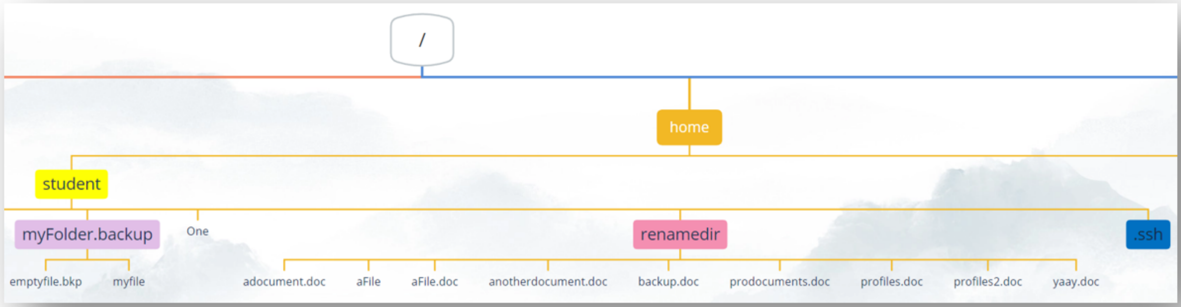
bash

```bash
student@linux-ess:~/renamedir$ ls
aFile  aFile.backup  adocument.txt  anotherdocument.txt  backup.txt  prod
student@linux-ess:~/renamedir$ rename 's/\.txt/.odt/' *
student@linux-ess:~/renamedir$ ls
aFile  aFile.backup  adocument.odt  anotherdocument.odt  backup.odt  prod
```

ℹ Notice how we put a `\` (backslash) in front of the `.` sign in the search-string? Some characters have special meanings in regular expressions (for example: `* . $ [ ] ( ) / { }`). If we want the bash shell to see this character as a string we have to use *escaping*. This is the concept of using the `\` to indicate that the character that follows is interpreted as a string rather than a special character.

We could also use the `rename` command to change multiple file extentions at once:





bash

```bash
student@linux-ess:~/renamedir$ ls
aFile  aFile.backup  adocument.odt  anotherdocument.odt  backup.odt  prod
student@linux-ess:~/renamedir$ rename -E 's/\.odt/.doc/i' -E 's/\.backup/
student@linux-ess:~/renamedir$ ls
aFile  aFile.doc  adocument.doc  anotherdocument.doc  backup.doc  prodocu
```

Notice how we put an `i` at the end of the perl expression to search case-insensitive. So the `rename -E 's/\.odt/.doc/i'` renames the odt extensions to doc regardless of the casing.

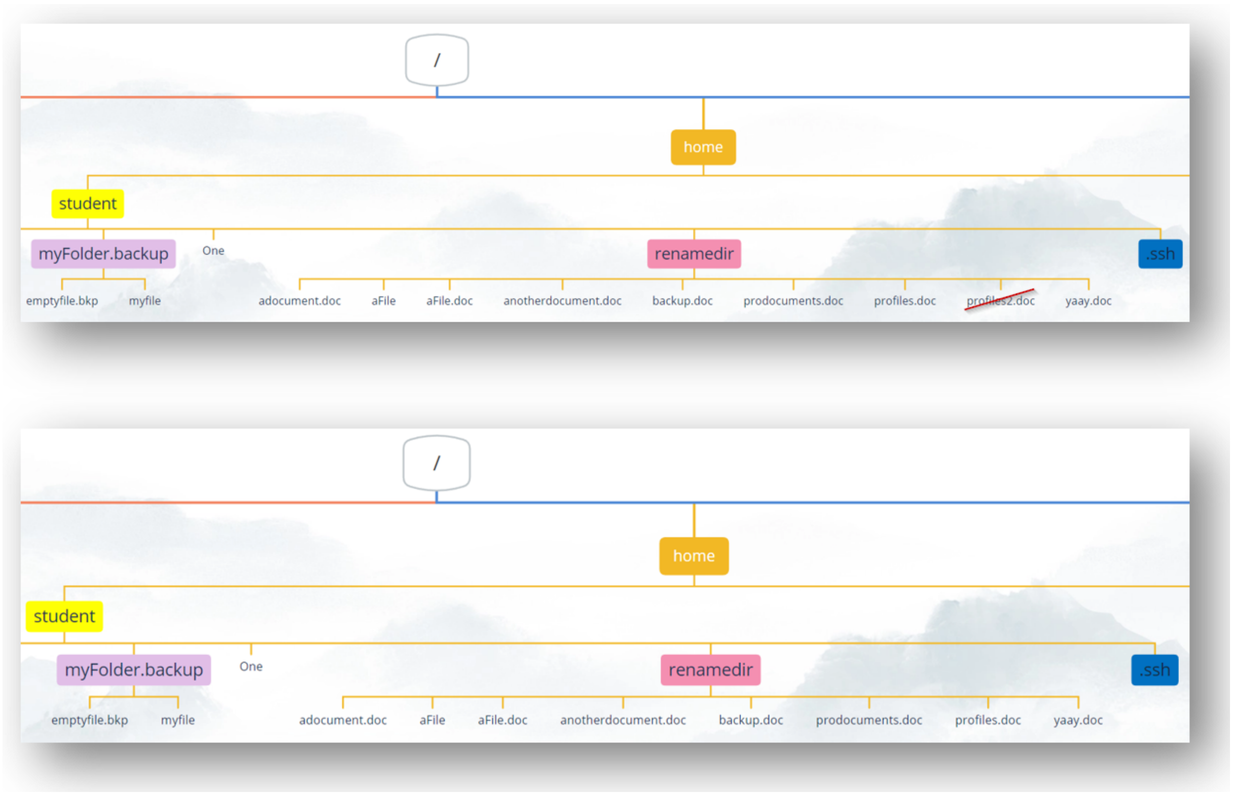
Identifying files (file)

In Linux we don't have to use file extentions. This means we don't always know the file type. We can use the `file` command to identify the type of a file:

```bash
student@linux-ess:~$ wget --cipher 'DEFAULT:!DH' http://www.pxl.be/img/lo
student@linux-ess:~$ file logo.png
logo.png: PNG image data, 150 x 150, 8-bit/color RGBA, non-interlaced
student@linux-ess:~$ file /etc/passwd
/etc/passwd: ASCII text
```

Delete files & folders (rm)

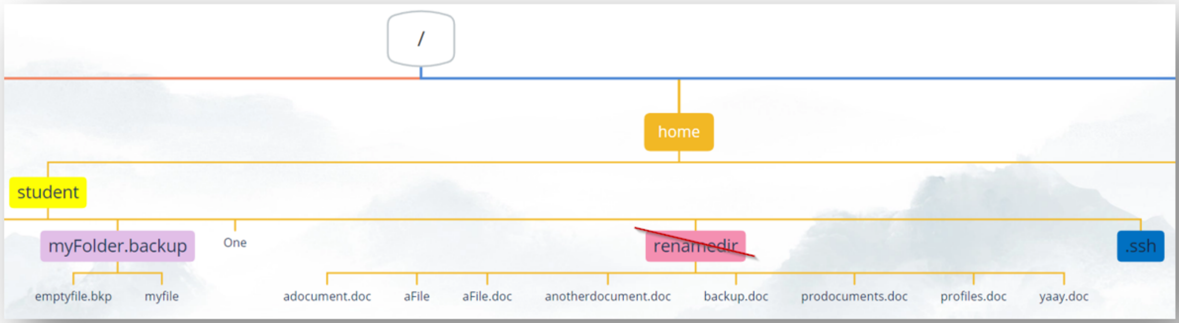
For deleting folders we could use the `rmdir` command but keep in mind that it wont delete folders containing other files or folders.

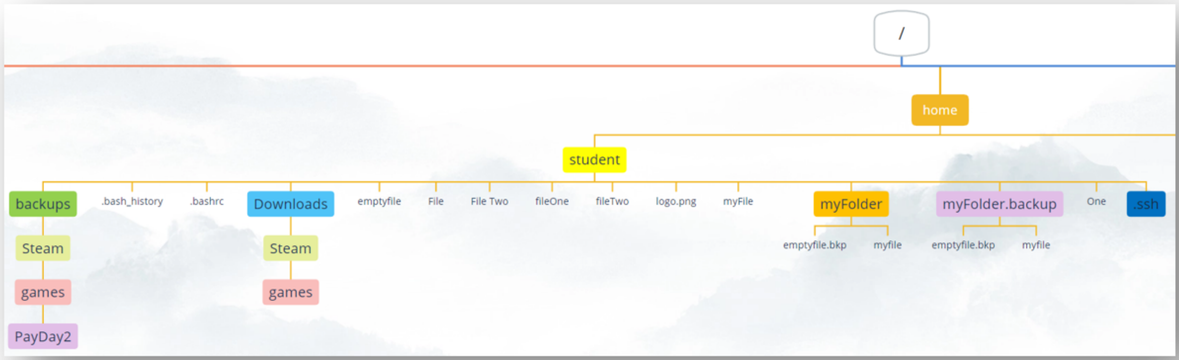


For deleting both files and folders we mostly use the `rm` command:

```bash
student@linux-ess:~/renamedir$ ls
aFile  aFile.doc  adocument.doc  anotherdocument.doc  backup.doc  prodocu
```

```
student@linux-ess:~/renamedir$ rm profiles2.doc
student@linux-ess:~/renamedir$ ls
aFile  aFile.doc  adocument.doc  anotherdocument.doc  backup.doc  prodocu
```





The `rm` command has different options as well, the most used combination is `rm -rf`:

- `-r` will mean it will remove files & folders recursive
- `-f` will force the command to remove non-empty directories as well. Something that wont happen out of the box.

bash

```
student@linux-ess:~$ ls
backups  Downloads  emptyfile  emptyfile.backup  File  fileOne  file
student@linux-ess:~$ ls renamedir
aFile  aFile.doc  adocument.doc  anotherdocument.doc  backup.doc  pro
student@linux-ess:~$ rm -rf renamedir
student@linux-ess:~$ ls
backups  Downloads  emptyfile  emptyfile.backup  File  fileOne  file
```

Be mindfull when using the `rm -rf` command as the root user!

There is no garbage bin in linux. Removing a file means its gone forever!

Searching for files

Searching on the filesystem (find)

The first command to search for files is `find`. It searches through the filesystem to locate files that we search for. We can specify in which directory the search starts (down the tree) and what the filename has to look like (with fileglobbing):

If you do not specify a directory the search begins in the current working directory. By default the search always continues down the tree into the subdirectories.

```bash
student@linux-ess:~$ find -name "*sh*"
./.bash_logout
./.local/share
./.lesshst
./.ssh
./.bashrc
./.bash_history
```

We can also specify a directory from where to start the search:

```bash
student@linux-ess:~$ find / -name "*networkmanager*"
find: ‘/boot/lost+found’: Permission denied
find: ‘/tmp/systemd-private-630e73882a5f4981b6052d8443c96a61-systemd-reso
find: ‘/tmp/systemd-private-630e73882a5f4981b6052d8443c96a61-systemd-logi
```

```
find: ‘/tmp/vmware-root_698-2730496923’: Permission denied
find: ‘/tmp/systemd-private-630e73882a5f4981b6052d8443c96a61-ModemManager
find: ‘/tmp/systemd-private-630e73882a5f4981b6052d8443c96a61-systemd-time
find: ‘/tmp/snap.lxd’: Permission denied
find: ‘/run/udisks2’: Permission denied
find: ‘/run/user/1000/systemd/inaccessible/dir’: Permission denied
find: ‘/run/sudo’: Permission denied
find: ‘/run/cryptsetup’: Permission denied
find: ‘/run/credentials/systemd-sysusers.service’: Permission denied
find: ‘/run/systemd/propagate’: Permission denied
find: ‘/run/systemd/unit-root’: Permission denied
find: ‘/run/systemd/inaccessible/dir’: Permission denied
find: ‘/run/lvm’: Permission denied
find: ‘/run/lock/lvm’: Permission denied
find: ‘/run/initramfs’: Permission denied
/usr/lib/python3/dist-packages/cloudinit/distros/parsers/networkmanager_c
/usr/lib/python3/dist-packages/cloudinit/distros/parsers/__pycache__/netw
/usr/lib/python3/dist-packages/sos/report/plugins/__pycache__/networkmana
/usr/lib/python3/dist-packages/sos/report/plugins/networkmanager.py
find: ‘/etc/polkit-1/localauthority’: Permission denied
find: ‘/etc/ssl/private’: Permission denied
...
```

We see too many error messages. It clutters the view and we don't get any results back from the directories where we have no privileges for the view their content. The solution is to use the `sudo` command:

bash

```bash
student@linux-ess:~$ sudo find / -name "*networkmanager*"
/usr/lib/python3/dist-packages/cloudinit/distros/parsers/networkmanager_c
/usr/lib/python3/dist-packages/cloudinit/distros/parsers/__pycache__/netw
/usr/lib/python3/dist-packages/sos/report/plugins/__pycache__/networkmana
/usr/lib/python3/dist-packages/sos/report/plugins/networkmanager.py
/snap/core20/1405/usr/lib/python3/dist-packages/cloudinit/distros/parsers
/snap/core20/1405/usr/lib/python3/dist-packages/cloudinit/distros/parsers
/snap/core20/1518/usr/lib/python3/dist-packages/cloudinit/distros/parsers
/snap/core20/1518/usr/lib/python3/dist-packages/cloudinit/distros/parsers
```

If we want our search to be case insensitive we can use `-iname` instead of `-name` :

bash

```
student@linux-ess:~$ sudo find / -iname "*networkmanager*"
/run/NetworkManager
/usr/lib/python3/dist-packages/cloudinit/distros/parsers/networkmanager_c
/usr/lib/python3/dist-packages/cloudinit/distros/parsers/__pycache__/netw
/usr/lib/python3/dist-packages/sos/report/plugins/__pycache__/networkmana
/usr/lib/python3/dist-packages/sos/report/plugins/networkmanager.py
/etc/NetworkManager
/snap/core20/1405/etc/NetworkManager
/snap/core20/1405/usr/lib/NetworkManager
/snap/core20/1405/usr/lib/python3/dist-packages/cloudinit/distros/parsers
/snap/core20/1405/usr/lib/python3/dist-packages/cloudinit/distros/parsers
/snap/core20/1518/etc/NetworkManager
/snap/core20/1518/usr/lib/NetworkManager
/snap/core20/1518/usr/lib/python3/dist-packages/cloudinit/distros/parsers
/snap/core20/1518/usr/lib/python3/dist-packages/cloudinit/distros/parsers
```

Searching through a filesystem database (locate)

Another way to search for files is to use the `locate` command. It searches through a database that holds information about the files in the filesystem.

> Before we use the command to search for the file(s) it is always good practice to update the database.

It's noticable that searching a database with `locate` is much faster than searching through the filesystem itself with `find. Make note that we do not specify a directory here. The search will always be done through all the files of the filesystem:

bash

```
student@linux-ess:~$ touch testfile
student@linux-ess:~$ ls
```

```
  emptyfile  emptyfile.backup  testfile
student@linux-ess:~$ find -name "testfile"
./testfile
student@linux-ess:~$ locate testfile
student@linux-ess:~$ sudo updatedb
student@linux-ess:~$ locate testfile
/home/student/testfile
student@linux-ess:~$ locate .profile
/etc/lvm/profile/cache-mq.profile
/etc/lvm/profile/cache-smq.profile
/etc/lvm/profile/command_profile_template.profile
/etc/lvm/profile/lvmdbusd.profile
/etc/lvm/profile/metadata_profile_template.profile
/etc/lvm/profile/thin-generic.profile
/etc/lvm/profile/thin-performance.profile
/etc/lvm/profile/vdo-small.profile
/etc/skel/.profile
/home/student/.profile
/snap/core20/1405/etc/skel/.profile
/snap/core20/1405/usr/share/base-files/dot.profile
...
```

If we also want to see files that we do not have the privileges for, then we have to use `sudo` :

```bash
student@linux-ess:~$ sudo locate .profile
/etc/lvm/profile/cache-mq.profile
/etc/lvm/profile/cache-smq.profile
/etc/lvm/profile/command_profile_template.profile
/etc/lvm/profile/lvmdbusd.profile
/etc/lvm/profile/metadata_profile_template.profile
/etc/lvm/profile/thin-generic.profile
/etc/lvm/profile/thin-performance.profile
/etc/lvm/profile/vdo-small.profile
/etc/skel/.profile
/home/student/.profile
/root/.profile
```

```
/snap/core20/1405/etc/skel/.profile
/snap/core20/1405/root/.profile
...
```

If we also want to search case insensitive, then we have to use the `-i` option:

```bash
student@linux-ess:~$ sudo locate -i networkmanager
/etc/NetworkManager
/etc/NetworkManager/dispatcher.d
/etc/NetworkManager/dispatcher.d/hook-network-manager
/snap/core20/1405/etc/NetworkManager
/snap/core20/1405/etc/NetworkManager/dispatcher.d
/snap/core20/1405/etc/NetworkManager/dispatcher.d/hook-network-manager
/snap/core20/1405/usr/lib/NetworkManager
/snap/core20/1405/usr/lib/NetworkManager/conf.d
/snap/core20/1405/usr/lib/NetworkManager/conf.d/no-mac-addr-change.conf
/snap/core20/1405/usr/lib/python3/dist-packages/cloudinit/distros/parsers
/snap/core20/1405/usr/lib/python3/dist-packages/cloudinit/distros/parsers
...
```

If we only want to search in filenames and not in directorynames we have to use the `-b` option:

```bash
student@linux-ess:~$ sudo locate -b -i networkmanager
/etc/NetworkManager
/snap/core20/1405/etc/NetworkManager
/snap/core20/1405/usr/lib/NetworkManager
/snap/core20/1405/usr/lib/python3/dist-packages/cloudinit/distros/parsers
/snap/core20/1405/usr/lib/python3/dist-packages/cloudinit/distros/parsers
/snap/core20/1518/etc/NetworkManager
/snap/core20/1518/usr/lib/NetworkManager
/snap/core20/1518/usr/lib/python3/dist-packages/cloudinit/distros/parsers
/snap/core20/1518/usr/lib/python3/dist-packages/cloudinit/distros/parsers
/usr/lib/python3/dist-packages/cloudinit/distros/parsers/networkmanager_c
/usr/lib/python3/dist-packages/cloudinit/distros/parsers/__pycache__/netw
```

```
/usr/lib/python3/dist-packages/sos/report/plugins/networkmanager.py
/usr/lib/python3/dist-packages/sos/report/plugins/__pycache__/networkmana
```

Extra course material

🎞 [Pluralsight] Linux command syntax patterns

🎞 [Pluralsight] Working with files & directories

Previous
3 Command line interface

Next
Lab