Lecture: Introduction

- Creating Files with `cat`
    - There are many ways of creating a file
    - One of the simplest is with the `cat` command:

```
$ cat > shopping_list
cucumber
bread
yoghurts
fish fingers
```

-
    - The text typed is written to a file with the specified name
    - Press Ctrl+D after a line-break to denote the end of the file
    - `ls` demonstrates the existence of the new file

```
$ls
```

- Displaying Files' Contents with `cat`
    - There are many ways of viewing the contents of a file
    - One of the simplest is with the `cat` command:

```
$ cat shopping_list
cucumber
bread
yoghurts
fish fingers
```

-
    - The text in the file is displayed immediately:

- Deleting Files with `rm`
    - To delete a file, use the rm ('remove') command
    - Simply pass the name of the file to be deleted as an argument:

```
$ rm shopping_list
```

-
    - The file and its contents are removed
    - There is no recycle bin
    - There is no 'unrm' command
    - The `ls` command can be used to confirm the deletion

- Copying and Renaming Files with `cp` and `mv`
    - To copy the contents of a file into another file, use the `cp` command:

```
$ cp /etc/hosts /home/student/hosts.bak
```

- To rename a file use the `mv` ('move') command:

```
$ mv /home/student/hosts.bak /home/student/Documents/hosts.bak
```

- If a file with the new name already exists, it is overwritten

Exercises

1.

a. Log in.
b. Log out.
c. Log in again. Open a terminal window, to start a shell.
d. Exit from the shell; the terminal window will close.
e. Start another shell. Enter each of the following commands in turn.
- date
- whoami
- hostname
- uptime

2.

a. Use the `ls` command to see if you have any files.

b. Create a new file using the `cat` command as follows:

```
$ cat > hello.txt
Hello world!
This is a text file.
```

Press Enter at the end of the last line, then Ctrl+D to denote the end of the file.

c. Use `ls` again to verify that the new file exists.

d. Display the contents of the file.

e. Display the file again, but use the cursor keys to execute the same command again without having to retype it.

3.

a. Create a second file. Call it *intrebari*, and put it wherever you want
b. Check its creation with `ls`.
c. Display the contents of this file. Minimise the typing needed to do this:

- Scroll back through the command history to the command you used to create the file.
- Change that command to display *intrebari* instead of creating it.

4. After each of the following steps, use **`ls`** and **cat** to verify what has happened.
   a. Copy *intrebari* to a new file called *raspunsuri*. Use Tab to avoid typing the existing file's name in full.
   b. Now copy *hello.txt* to *raspunsuri*. What's happened now?
   c. Delete the original file, *hello.txt*.
   d. Rename *raspunsuri* to *message*.
   e. Try **`rm`** to delete a file called *missing*. What happens?
   f. Try copying *intrebari* again, but don't specify a filename to which to copy.
   What happens now?

Lecture: Basic file management

- Filename Completion
  - Modern shells help you type the names of files and directories by completing partial names
  - Type the start of the name (enough to make it unambiguous) and press **Tab**
  - For an ambiguous name (there are several possible completions), type **Tab** twice in succession

- Making Directories with **mkdir**
  - Syntax: mkdir <directory_name>
  - Options:
  - -p, create intervening parent directories if they don't already exist
  - -m mode, set the access permissions to mode
  - For example, create a directory called mystuff in your home directory with permissions so that only you can write, but eveyone can read it:

```
$ mkdir -m 755 ~/mystuff
```

  - Create a directory tree in /tmp using one command with three subdirectories called one, two and three:

```
$ mkdir -p /tmp/one/two/three
```

- Removing Directories with **rmdir**
  - rmdir deletes empty directories, so the files inside must be deleted first
  - For example, to delete the **two** directory from the /tmp/one/two/three path:

```
$ rm three
$ rmdir two
```

- For non-empty directories, use rm -r directory
- The -p option to rmdir removes the complete path, if there are no other files and directories in it
- These commands are equivalent:

```
$ rmdir -p /tmp/one
$ rmdir /tmp/one/two/three /tmp/one/two /tmp/one
```

<u>Basic regular expressions</u>

- A *regular expression* is a pattern that describes a set of strings.
- A *meta character* is one or more special characters that have a unique meaning and are NOT used as literals in the search expression, for example, the character ^ (circumflex or caret) is a meta character

Most used meta characters:

^ – Caret/Power symbol to match a starting at the beginning of line.
$ – To match end of the line
* – To match any character
? – To match any single character
[] – Range of character
- – The - (dash) inside square brackets is the 'range separator' and allows us to define a range, in our example above of [0123456789] we could rewrite it as [0-9].
You can define more than one range inside a list, for example, [0-9A-C] means check for 0 to 9 and A to C (but not a to c).
[^char] – negate of occurrence of a character set
/ (or ./) - will be used when you want to manage special characters. For example: ./- will be interpreted as minus sign character

- Create some files ending in .txt
- Use the symbol * in **ls** command to match any part of a filename in your current working directory

```
$ ls *.txt
```

- Just ***** produces the names of all files in the current directory

```
$ ls *
```

- The wildcard **?** matches exactly one character. Create the files *data.1*, *data.2*, *data.33* then list only the files with only one digit after .

```
$ ls data.?
```

- Copying Files with cp
    - Syntax: cp [options] source-file destination-file
    - Copy multiple files into a directory: cp <file> <directory>
    - Common options:
        - -f, force overwriting of destination files
        - -i, interactively prompt before overwriting files
        - -a, archive, copy the contents of directories recursively
    - Examples of cp usage:
        - Copy /etc/sysconfig/network to the current directory:

```
$ cp /etc/sysconfig/network .
```

- Create an identical copy of a directory called Documents, and call it Documents-backup:

```
$ cp -a Documents Documents-backup
```

- Moving Files with mv
    - mv can rename files or directories, or move them to different directories
    - It is equivalent to copying and then deleting
    - But is usually much faster
    - Options:
        - -f, force overwrite, even if target already exists
        - -i, ask user interactively before overwriting files
    - For example, to rename poetry.txt to poems.txt:
        - Create poetry.txt first, then

```
$ mv poetry.txt poems.txt
```

- To move everything in the current directory somewhere else:
    - Create old-stuff directory first, then

```
$ mv * ~/old-stuff/
```

- Deleting Files with rm
    - rm deletes ('removes') the specified files

- You must have write permission for the directory the file is in to remove it
- Use carefully if you are logged in as root!
- Options:
  - -f, delete write-protected files without prompting
  - -i, interactive — ask the user before deleting files
  - -r, recursively delete files and directories

- For example, clean out everything in /tmp, without prompting to delete each file:

```
$ rm -rf /tmp/*
```

- Deleting files with peculiar names
  - Create a file and name it –filename

```
$touch –filename
```

  - Going to give the command as above it will give you an error, because the minus signed is treated as a begining for an option on **touch** command, not as a character

```
$ touch ./-filename
```

- Identifying Types of Files
  - The data in files comes in various different formats (executable programs, text files, etc.)
  - The **file** command will try to identify the type of a file:

```
$ file /bin/bash
/bin/bash: ELF 32-bit LSB executable, Intel 80386, version 1,
dynamically linked (uses shared libs), stripped
```

  - Useful to find out whether a program is actually a script:

```
$ file /usr/bin/zless
```

  - Try this file types also:

```
$file /etc
$file /etc/host
$file /bin/ls
$file /usr/share/pixmaps/gnome-gimp.png
```

Case:

- List all the directories found on root level (the /)

```
$ls -ld /*
```

- list all the files in */etc* directory with 3 characters extension

```
$ls /etc/*.???
```

- List all the files from root which ends in *r* (like /usr)

```
$ls -d /??r
```

- Create the following files: *file11.txt file63.txt file71.txt file42.txt*
- Do some practice using **ls** command with meta characters.

```
$ls -l f*[1]*
-rw-r--r-- 1 root root 0 Oct 26 16:23 file11.txt
-rw-r--r-- 1 root root 0 Oct 26 16:23 file71.txt

$ls -l f*[12]1*
-rw-r--r-- 1 root root 0 Oct 26 16:23 file11.txt

$ls -l file[1-5]*
-rw-r--r-- 1 root root 0 Oct 26 16:23 file11.txt
-rw-r--r-- 1 root root 0 Oct 26 16:23 file42.txt
```

  - Negate: use **!** and ^ to get anything but not what is between []

```
$ ls -l file[!1-5]*
-rw-r--r-- 1 root root 0 Oct 26 16:23 file63.txt
-rw-r--r-- 1 root root 0 Oct 26 16:23 file71.txt

$ls -l /dev/[hs]da[1-4]
```

Exercises

1.

a. Use cd to go to your home directory, and create a new directory there called *dog*.
b. Create another directory within that one called *cat*, and another within that called *mouse*.
c. Remove all three directories. You can either remove them one at a time, or all at once.
d. If you can delete directories with rm -r, what is the point of using rmdir for empty directories?
e. Try creating the *dog/cat/mouse* directory structure with a single command.

2.

a. Copy the file */etc/passwd* to your home directory, and then use cat to see what's in it.

b. Rename it to *users* using the mv command.
c. Make a directory called *programs* and copy everything from */bin* into it.
d. Delete all the files in the *programs* directory.
e. Delete the empty *programs* directory and the *users* file.

3.

a. The touch command can be used to create new empty files. Try that now, picking a name for the new file:

```
$ touch baked-beans
```

b. Get details about the file using the ls command:

```
$ ls -l baked-beans
```

c. Wait for a minute, and then try the previous two steps again, and see what changes. What happens when we don't specify a time to touch?
d. Try setting the timestamp on the file to a value in the future.
e. When you're finished with it, delete the file.

4.

a. Use the pwd command to find out what directory you are in.
b. If you are not in your home directory (/home/USERNAME) then use cd without any arguments to go there, and do pwd again.
c. Use cd to visit the root directory, and list the files there. You should see **home** among the list.
d. Change into the directory called home and again list the files present. There should be one directory for each user, including the user you are logged in as (you can use **`whoami`** to check that).
e. Change into your home directory to confirm that you have gotten back to where you started.

5.

a. Create a text file in your home directory called **Eminescu**, containing the following text:

```
A fost odata ca-n povesti
A fost ca niciodata...
```

b. Rename it to **luceafarul.txt**.
c. Make a new directory in your home directory, called **poezie**.
d. Move **Eminescu** into the new directory.

e. Try to find a graphical directory-browsing program (ex. **Nautilus**), and find your home directory with it. You should also be able to use it to explore some of the system directories.
f. Find a text editor program and use it to display and edit **Eminescu**.

6.

a. From your home directory, list the files in the directory **/usr/share**.
b. Change to that directory, and use pwd to check that you are in the right place. List the files in the current directory again, and then list the files in the directory called **doc**.
c. Next list the files in the parent directory, and the directory above that.
d. Try the following command, and make sure you understand the result:

```
$ echo ˜
```

e. Use cat to display the contents of a text file which resides in your home directory (create one if you haven’t already), using the ˜/ syntax to refer to it. It shouldn’t matter what your current directory is when you run the command.

7.

a. Use the **hostname** command, with no options, to print the hostname of the machine you are using.
b. Use **man** to display some documentation on the hostname command. Find out how to make it print the IP address of the machine instead of the hostname. You will need to scroll down the manpage to the ‘Options’ section.
c. Use the **locate** command to find files whose name contains the text ‘hostname’. Which of the filenames printed contain the actual hostname program itself? Try running it by entering the program’s absolute path to check that you really have found it.

8.

a. The * wildcard on its own is expanded by the shell to a list of all the files in the current directory. Use the **`echo`** command to see the result (but make sure you are in a directory with a few files or directories first)
b. Use quoting to make echo print out an actual * symbol.
c. Create in the directory **poezie** you created earlier another file, **lacul.txt**:

```
Lacul codrilor albastru
Nuferi galbeni îl încarcă;
```

d. Use the **`cat`** command to display both of the poems, using a wildcard.
e. Finally, use the **`rm`** command to delete **poezii** directory and the poems in it.

<u>Chaining programs together: uing pipes and redirects</u>

- A pipe channels the output of one program to the input of another
- Allows programs to be chained together
- Programs in the chain run concurrently
- Use the vertical bar: |, sometimes known as the 'pipe' character
- For example, pipe the output of echo into the program rev (which reverses each line of its input):

```
$ echo Happy Birthday! | rev
!yadhtriB yppaH
```

- The **who** command lists the users currently logged in
- The **wc** command counts bytes, words, and lines in its input
- We combine them to count how many users are logged in:

```
$ who | wc –l
```

  - The | symbol makes a **pipe** between the two programs
  - The output of **who** is fed into **wc**
  - The **-l** option makes **wc** print only the number of lines
  - Another example, to join all the text files together and count the words, lines and characters in the result (you need to have some .txt files created, first):

```
$ cat *.txt | wc
```

- I/O Redirect categories:
  - Standard output
    - Most command line programs that display their results do so by sending their results to a facility called standard output. By default, standard output directs its contents to the display. To redirect standard output to a file, the „>" character is used like this:

```
$ls > filelist
```

    - In this example, the **ls** command is executed and the results are written in a file named filelist. Since the output of **ls** was redirected to the file, no results appear on the display.
    - Each time the command above is repeated, filelist is overwritten (from the beginning) with the output of the command **ls**. If you want the new results to be appended to the file instead, use „>>" like this:

```
$ls >> filelist
```

        - When the results are appended, the new results are added to the end of the file, thus making the file longer each time the command is repeated. If the file does not exist when you attempt to append the redirected output, the file will be created.

    - Standard input
        - Many commands can accept input from a facility called standard input. By default, standard input gets its contents from the keyboard, but like standard output, it can be redirected. To redirect standard input from a file instead of the keyboard, the „<” character is used like this:

```
$sort < filelist
```

        - In the above example we used the sort command to process the contents of *filelist*. The results are output on the display since the standard output is not redirected in this example. We could redirect standard output to another file like this:

```
$sort < filelist > sorted_filelist
```

<u>Case:</u>

Create a directory called *dir1*, then try create another one with the same name, You should get an error. The errors received can be redirected to specific files, using „2>” descriptor:

```
$ mkdir dir1 2> error.log
```

Exercises

1.

a. Try the example on the 'Pipes' slide, using rev to reverse some text.
b. Try replacing the echo command with some other commands which produce output (e.g., whoami).
c. What happens when you replace rev with cat? You might like to try running cat with no arguments and entering some text.

2.

a. Run the command ls --color in a directory with a few files and directories. Some Linux distributions have ls set up to always use the --color option in normal circumstances, but in this case we will give it explicitly.
b. Try running the same command, but pipe the output into another program (e.g., cat or less). You should spot two differences in the output. ls detects whether its output is going straight to a terminal (to be viewed by a human directly) or into a pipe (to be read by another program).