November 12, 2025

NUMERICAL METHODS I

Lab 1: Linux Operating System, Unix Commands, Bash shell

Hervé Tajouo Tela

United Nations Educational, Scientific and Cultural Organization
ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

What is an Operating System (OS)?

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

- Software interface between the user and the computer hardware
- Controls the execution of other programs
- Responsible for managing multiple computer resources (CPU, memory, disk, display, keyboard, etc.)
- Examples of OS: Windows, Unix/Linux, OSX.

Unix Introduction

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

- UNIX is an operating system which was first developed in the 1960s and has been under constant development since.
- It is a stable, multi-user, multi-tasking system for servers, desktops and laptops.
- UNIX systems also have a graphical user interface (GUI) similar to Microsoft Windows which provides an easy to use environment.
- There are many different versions of UNIX, although they share common similarities. The most popular varieties of UNIX are Sun Solaris, GNU/Linux and MacOS X.

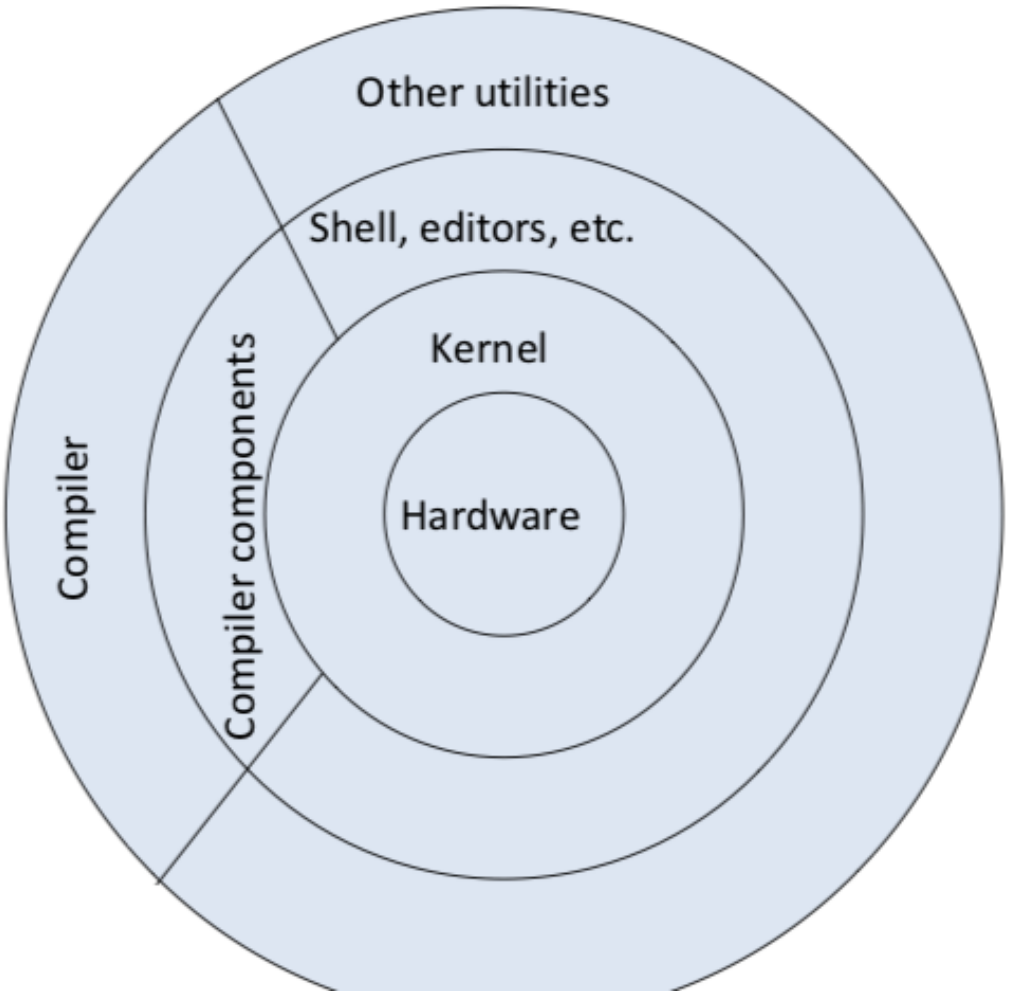
The UNIX operating system

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

- The UNIX operating system is made of 3 parts: the kernel, the shell and the programs.
- The kernel of UNIX is the hub of the operating system: it allocates time and memory to programs and handles the filestore and communications in response to system calls.
- The shell acts as an interface between the user and the kernel. When a user logs in, the login program checks the username and password, and then starts another program called the shell. The shell is a command line interpreter (CLI).

How does the Linux OS Work?



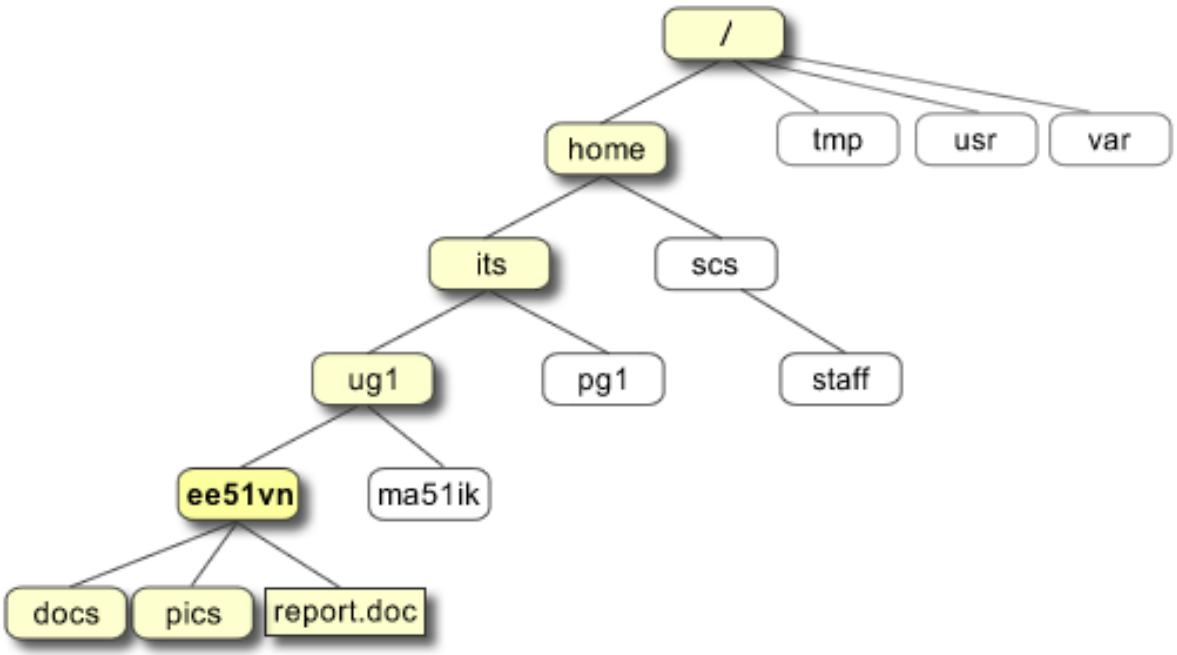
- Linux has a kernel and one or more shells
- The shell is the command line interface through which the user interacts with the OS. Most commonly used shell is “bash”
- The kernel sits on top of the hardware and is the core of the OS; it receives tasks from the shell and performs them

Linux File System

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

- A directory in Linux is similar to a "Folder" in Windows OS
- Files are organized into directories and sub-directories.
- In Linux, paths begin at the root directory which is the top-level of the file system and is represented as a forward slaxh (/).
- Forward slash is used to separate directory and file names.



The Command Line: Basic Navigation

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

A command line or terminal is a text based interface to the system. You are able to enter commands by typing them on the keyboard and feedback will be given to you as text.

Where are we?

- `pwd`: "Print Working Directory".
  It tells you what your current or present directory is.

```
Terminal
1. user@bash: ls
2. bin Documents public_html
3. user@bash:
```

What's in our current location?

- `ls`: "List" ⇒ `ls [options][location]`
  It tells you what is in your current or present directory.

```
Terminal
1. user@bash: ls
2. bin Documents public_html
3. user@bash:
```

The Command Line: Basic Navigation

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

What's in our current location?

- `ls`: "List" ⇒ `ls [options][location]`
  It tells you what is in your current or present directory.

Terminal

```
1.  user@bash: ls
2.  bin Documents public_html
3.  user@bash:
4.  user@bash: ls -l
5.  total 3
6.  drwxr-xr-x  2 ryan users 4096 Mar 23 13:34 bin
7.  drwxr-xr-x 18 ryan users 4096 Feb 17 09:12 Documents
8.  drwxr-xr-x  2 ryan users 4096 May 05 17:25 public_html
9.  user@bash:
10. user@bash: ls /etc
11. a2ps.cfg aliases alsa.d cups fonts my.conf systemd
12. ...
13. user@bash: ls -l /etc
14. total 3
15. -rwxr-xr-x  2 root root 123 Mar 23 13:34 a2ps.cfg
16. -rwxr-xr-x 18 root root 78 Feb 17 09:12 aliases
17. drwxr-xr-x  2 ryan users 4096 May 05 17:25 alsa.d
18. ...
19. user@bash:
```

The Command Line: Basic Navigation

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Absolute and relative paths

A relative path is about a file or directory location relative to where we currently are in the file system. An absolute path is about a file or directory location in relation to the root of the file system.

- `pwd`: "Print Working Directory".
- `ls`: "List" $\Rightarrow$ `ls [options][location]`

Terminal

```
1.  user@bash: pwd
2.  /home/ryan
3.  user@bash:
4.  user@bash: ls Documents
5.  file1.txt file2.txt file3.txt
6.  ...
7.  user@bash: ls /home/ryan/Documents
8.  file1.txt file2.txt file3.txt
9.  ...
10. user@bash:
```

The Command Line: Basic Navigation

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

More on paths

- ~(tilde): this is a shortcut for your home directory. eg, if your home directory is `/home/steve` then you could refer to the directory `Documents` with the path `/home/steve/Documents` or `~/Documents`.
- .(dot): this is a reference to your current directory. eg, in the previous example we referred to `Documents` on line 4 with a relative path. It could also be written as `./Documents`.
- ..(dotdot): this is a reference to the parent directory. You can use this several times in a path to keep going up the hierarchy. eg if you were in the path `/home/steve` you could run the command `ls ../../` and this would do a listing of the root directory.

The Command Line: Basic Navigation

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

More on paths

- `pwd`: "Print Working Directory".
- `ls`: "List" ⇒ `ls [options][location]`

Terminal

```
1.  user@bash: pwd
2.  /home/ryan
3.  user@bash:
4.  user@bash: ls ~/Documents
5.  file1.txt file2.txt file3.txt
6.  ...
7.  user@bash: ls ./Documents
8.  file1.txt file2.txt file3.txt
9.  ...
10. user@bash: ls /home/ryan/Documents
11. file1.txt file2.txt file3.txt
12. ...
13. user@bash:
14. user@bash: ls ../../
15. bin boot dev etc home lib var
16. ...
17. user@bash:
18. user@bash: ls /
19. bin boot dev etc home lib var
20. ...
```

The Command Line: Basic Navigation

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Let's move around a bit

- `pwd`: "Print Working Directory".
- `ls`: "List" ⇒ `ls [options][location]`
- `cd`: "Change Directory" ⇒ `cd [location]`

Terminal

```
1.  user@bash: pwd
2.  /home/ryan
3.  user@bash: cd Documents
4.  user@bash: ls
5.  file1.txt file2.txt file3.txt
6.  ...
7.  user@bash: cd /
8.  user@bash: pwd
9.  /
10. user@bash: ls
11. bin boot dev etc home lib var
12. ...
13. user@bash: cd ~/Documents
14. user@bash: pwd
15. /home/ryan/Documents
16. user@bash: cd ../../
17. user@bash: pwd
18. /home
19. user@bash: cd
20. user@bash: pwd
21. /home/ryan
```

Basic Navigation: Exercises

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

- Use the commands cd and ls to explore what directories are on your system and what's in them. Make sure you use a variety of relative and absolute paths. Some interesting places to look at are:
  - /etc – Stores config files for the system.
  - /var/log – Stores log files for various system programs. (You may not have permission to look at everything in this directory. Don't let that stop you exploring though. A few error messages never hurt anyone.)
  - /bin – The location of several commonly used programs (some of which we will learn about in the rest of this tutorial.
  - /usr/bin – Another location for programs on the system.
- Now go to your home directory using 4 different methods..
- Make sure you are using Tab Completion when typing out your paths too. Why do anything you can get the computer to do for you?

More About Files

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Everything is a file:

Everything in linux is a file. A text file is a file, a directory is a file, your keyboard is a file, your monitor is a file, etc.

Linux is case sensitive:

Unlike Windows which is case insensitive, in Linux it is possible to have two or more files and directories with the same name but letters of different case.

Linux is an extensionless system:

In Linux unlike Windows, the system ignores the extension of a file and looks inside the file to determine what type of file it is.

- `file`: "type of file" $\Rightarrow$ `file [path]`

Terminal

```
1. user@bash: ls Documents
2. FILE1.txt File1.txt file1.TXT
3. ...
4. user@bash: file Documents/file1.txt
5. Documents/file1.txt: ERROR: cannot open 'file1.txt' (No such file or
   directory)
```

More About Files

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Space in names:

A space on the command line is how we separate items. They are how we know about what is the program name and can identify each command line argument. Space in files are valid but we need to be careful with them.

Terminal

```
1. user@bash: ls Documents
2. FILE1.txt File1.txt file1.TXT Holiday Photos
3. ...
4. user@bash: cd Holiday Photos
5. bash: cd: Holiday: No such file or directory
```

Quotes:

To represent space in names we can use quotes around the entire item. Anything inside quotes is considered a single item.

Terminal

```
1. user@bash: cd 'Holiday Photos'
2. user@bash: pwd
3. /home/ryan/Documents/Holiday Photos
```

More About Files

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Escape characters:

Another way of representing spaces in names is to use and escape character, which is a backslash (\).

Terminal

```
1. user@bash: cd Holiday\ Photos
2. user@bash: pwd
3. /home/ryan/Documents/Holiday Photos
```

Hidden files and directories:

In Linux, files or directories hat are hidden have their name beginning with a .(full stop). Files are hidden for various reasons: configuration files are usually hidden. To view hidden files and directories in your current location you simply have to type `ls -a`.

Terminal

```
1. user@bash: ls Documents
2. FILE1.txt File1.txt file1.TXT
3. ...
4. user@bash: ls -a Documents
5. . .. FILE1.txt File1.txt file1.TXT .hidden .file.txt
6. ...
```

More About Files: Exercises

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

- Try running the command **file** giving it a few different entries. Make sure you use a variety of absolute and relative paths when doing this.

- Now issue a command that will list the contents of your home directory including hidden files and directories.

Manual Pages

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

What are manual pages?

Manual pages are a set of pages that explain every command available on your syetem including what they do, the specifics of how to run them and what command line arguments they accept. To invoke the manual pages you should type the following command: `man <command to look up>`.

Terminal

```
user@bash: man ls
Name
    ls - list directory contents

Synopsis
    ls [option] ... [file] ...

Description
    List information about the FILEs (the current directory by default). Sort
entries alphabetically if none of -cftuvSUX nor --sort is specified.

    Mandatory arguments to long options are mandatory for short options too.

    -a, --all
        do not ignore entries starting with .

    -A, --almost-all
        do not list implied . and ..

...
```

Manual Pages

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Searching the manual pages

It is possible to do a keyword search on the Manual Pages: it is helpful if you're not qure about what command to use but you know what you want to achieve. To invoke the manual pages search you should type the following command: `man -k <search term>`. While you are in a manual page you can also perform a search by pressing `'/'` followed by the term you want to search. You can go through all the possible options by pressing `n`.

More on the Running of Commands

Being proficient at Linux often means knowing which command line options we should use to modify the behaviour of our commands to suit our needs. Most of these commands have a long and short version: the long version is often easier to remember but the short one allows you to chain multiple together more easily.

Terminal

```
1. user@bash: pwd
2. /home/ryan
3. user@bash: ls -a
4. user@bash: ls --all
5. user@bash: ls -alh
6. user@bash:
```

Manual Pages: Exercises

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

- Have a skim through the man page for **ls**. Have a play with some of the command line options you find there. Make sure you play with a few as combinations. Also make sure you play with **ls** with both absolute and relative paths.

- Now try doing a few searches through the man pages. Depending on your chosen terms you may get quite a large listing. Look at a few of the pages to get a feel for what they are like..

File Manipulation

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Making a directory

Making a directory is pretty easy. The command is simply

- `mkdir`: "Make Directory" ⇒ `mkdir [options] <Directory>`

Terminal

```
1.  user@bash: pwd
2.  /home/ryan
3.  user@bash:
4.  user@bash: ls
5.  bin Documents public_html
6.  user@bash:
7.  user@bash: mkdir linuxtutorialwork
8.  user@bash:
9.  user@bash: ls
10. bin Documents linuxtutorialwork public_html
```

Remember that when we supply a directory in the above comand we are actually supplying a path which can be a relative or absolute path.

File Manipulation

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Making a directory

There are a few useful options available for `mkdir`. The option `-p` which tells `mkdir` to make parent directories as needed. The option `-v` which makes `mkdir` tell us what it does.

Terminal

```
1. user@bash: mkdir -p linuxtutorialwork/foo/bar
2. user@bash:
3. user@bash: cd linuxtutorialwork/foo/bar
4. user@bash: pwd
5. /home/ryan/linuxtutorialwork/foo/bar
```

Terminal

```
1. user@bash: mkdir -pv linuxtutorialwork/foo/bar
2. mkdir: created directory 'linuxtutorialwork/foo'
3. mkdir: created directory 'linuxtutorialwork/foo/bar'
4. user@bash:
5. user@bash: cd linuxtutorialwork/foo/bar
6. user@bash: pwd
7. /home/ryan/linuxtutorialwork/foo/bar
```

File Manipulation

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Removing a directory

Removing or deleting a directory is also easy. However there is no undo, so one has to be careful before using this command. The command is simply

- `rmdir`: "Remove Directory" ⇒ `rmdir [options] <Directory>`

Terminal

```
1. user@bash: rmdir linuxtutorialwork/foo/bar
2. user@bash:
3. user@bash: ls linuxtutorialwork/foo
4.
```

`rmdir` supports the options `-p` and `-v` similar to `mkdir`.

Also the directory must be empty before it may be removed.

File Manipulation

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Creating a Blank File

A lot of commands that involve manipulating data within a file have the feature that they will create a file automatically if we refer to it and it doesn't exist. We can use this charateristic to create blank files using a command

- `touch`: ⇒ `touch [options] <filename>`

Terminal

```
1.  user@bash: pwd
2.  /home/ryan/linuxtutorialwork
3.  user@bash:
4.  user@bash: ls
5.  foo
6.  user@bash:
7.  user@bash: touch example1
8.  user@bash:
9.  user@bash: ls
10. example1 foo
```

File Manipulation

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Copying a File or Directory

The command to copy a file or directory is simple

- cp: "Copy" ⇒ `cp [options] <source> <destination>`

Terminal

```
1. user@bash: ls
2. example1 foo
3. user@bash:
4. user@bash: cp example1 barney
5. user@bash: ls
6. barney example1 foo
```

Terminal

```
1. user@bash: ls
2. barney example1 foo
3. user@bash: cp foo foo2
4. cp: omitting directory 'foo'
5. user@bash: cp -r foo foo2
6. user@bash: ls
7. barney example1 foo foo2
```

`cp` supports various options among which `-r` ('recursive') allows to copy directories.

File Manipulation

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Moving a File or Directory

Moving files or directories is done with a command similar to `cp` with the advantage that we can me move directories without having to provide the `-r` option. The command is simply

- `mv`: "Move" ⇒ `mv [options] <source> <destination>`

Terminal

```
1. user@bash: ls
2. barney example1 foo foo2
3. user@bash: mkdir backups
4. user@bash: mv foo2 backups/foo3
5. user@bash: mv barney backups/
6. user@bash: ls
7. backups example1 foo
```

`rmdir` supports the options `-p` and `-v` similar to `mkdir`.

Also the directory must be empty before it may be removed.

File Manipulation

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Renaming Files and Directories

We can use the command `mv` in a creative way to achieve the outcome of renaming a file or directory. If we specify the destination to be the same directory as the source, but with a different name, then we have used `mv` to rename a file or directory.

Terminal

```
1.  user@bash: ls
2.  backups example1 foo
3.  user@bash: mv foo foo3
4.  user@bash: ls
5.  backups example1 foo3
6.  user@bash: cd ..
7.  user@bash: mkdir linuxtutorialwork/testdir
8.  user@bash: mv linuxtutorialwork/testdir /home/ryan/linuxtutorialwork/fred
9.  user@bash: ls linuxtutorialwork
10. backups example1 foo3 fred
```

File Manipulation

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Removing a File

Just like with `rmdir`, the action of removing a file can't be undone, so one has to be careful before using this command. The command is simply

- `rm`: "Remove" ⇒ `rm [options] <file>`

Terminal

```
1. user@bash: ls
2. backups example1 foo3 fred
3. user@bash: rm example1
4. user@bash: ls
5. backups foo3 fred
```

File Manipulation

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Removing non empty Directories

`rm` has several options that alter its behaviour. The option `-r` is a particularly useful one and it behaves in a similar fashion as with `cp`. It allows us to remove directories and all files and directories contained within.

Terminal

```
1. user@bash: ls
2. backups foo3 fred
3. user@bash: rmdir backups
4. rmdir: failed to remove 'backups': Directory not empty
5. user@bash: rm backups
6. rm: cannot remove 'backups': Is a directory
7. user@bash: rm -r backups
8. user@bash: ls
9. foo3 fred
```

A combination of options is mixing the `-r` with `-i` ('interactive') as it will prompt you before removing each file and directory and it gives you the option to cancel the command.

File Manipulation: Exercises

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

- Start by creating a directory in your home directory in which to experiment.
- In that directory, create a series of files and directories (and files and directories in those directories).
- Now rename a few of those files and directories.
- Delete one of the directories that has other files and directories in them.
- Move back to your home directory and from there copy a file from one of your subdirectories into the initial directory you created.
- Now move that file back into another directory.
- Rename a few files
- Next, move a file and rename it in the process.
- Finally, have a look at the existing directories in your home directory. You probably have a 'Documents', 'Downloads', 'Music' and 'Images' directory etc. Think about what other directories may help you keep your account organised and start setting this up.

Vi Text Editor

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

A Command Line Editor

- `vi` is a command line text editor.
- `vi` has been designed to work with the limitations of being a single window with text input and output only, just like the command line.
- `vi` is intended as a plain text editor (similar to `Notepad` on Windows or `Textedit` on Mac) as opposed to a word processing suite such as `Word` or `Pages`.
- Everything in `vi` is done with the keyboard.
- `vi` has two modes: `insert` (or input) mode and `edit` mode.
- In the `input` mode you may input or enter content into the file.
- In the `edit` mode you can move around the file, perform actions such as deleting, copying, search and replace, saving etc.

Vi Text Editor

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

`vi` is usually use with a simple command: `vi <file>`. It allows us to remove directories and all files and directories contained within.

Terminal

```
1. user@bash: vi firstfile
```

Once the file is open you are in `edit` mode.

Terminal

```
1. ~
2. ~
3. ~
4. ~
5. ~
6. "firstfile" [New File]
```

To get into `insert` mode you can press the `i`.

Terminal

```
1. ~
2. ~
3. ~
4. ~
5. ~
6. -- INSERT --
```

After editing the file you can press `Esc` and return to `edit` mode.

Vi Text Editor

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Saving and Exiting

There are several ways about doing this. They all essentially do the same thing so pick whichever way youp prefer. However, for all these, make sure you are in `edit` mode first.

- `ZZ` (in capitals) – save and exit.
- `:q!` – discard all changes since the last save and exit.
- `:w` – save file but don't exit.
- `:wq` – again, save and exit.
- `:x` – once more, save and exit.

Vi Text Editor

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Other Ways to View Files

`vi` allows us to edit files. But it can also be used to view files. If we simply want to view a file, a few commands are available to do it.

- The command `less` allows yo to view large files. `less <file>`. `less` allows you to move up and down within a file using the arrow keys. You may go forward a whole page using the `SpaceBar` and quit using `q`. A command similar to `less` is the command `more`.
- The command `cat` ('concatenate') has as its main purpose to join files together. But it can also be used just to view files. It is used as `cat <file>`.

Terminal

```
1. user@bash: cat firstfile
2. here you will see
3. whatever content you
4. entered in your file
5. user@bash:
```

However if the file is large, then most of the content will fly accross the screen and we'll only see the last page. The commands `less` and `more` will therefore be more appropriate to view large files.

Vi Text Editor

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Navigating a file in vi

Once you have opened a file using the `vi` command, in `insert` mode you can use the arrow keys to move the cursor around. Then hitting the `Esc` button you may go back to `edit` mode. Here are some commands that that cand be entered to move around a file.

- `Arrow keys` – move the cursor around.
- `j, k, h, l` – move the cursor down, up, left and right (similar to the arrow keys).
- `^(caret)` – move cursor to the beginning of current line.
- `$` – move cursor to the end of the current line.
- `nG` – move to the `n`th line.
- `G` – move to the last line.
- `w` – move to the beginning of the word.
- `nw` – move forward `n` word.
- `b` – move to the beginning of the previous word.
- `w` – move back `n` word.
- `{` – move backward one paragraph.
- `}` – move forward one paragraph.

Vi Text Editor

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Deleting Content

Here are some commands that can be entered to delete content in `vi`.

- `x` – delete a single character.
- `nx` – delete n characters.
- `dd` – delete the current line.
- `dn` – d followed by a movement command. Delete to where the movement command would have taken you. (eg d3w means delete 5 words).

Undoing

Here are 2 commands that can be entered to undo and action in `vi`.

- `u` – undo the last action.
- `U (capital)` – undo all the changes to the current line.

Vi Text Editor: Exercises

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

- Start by creating a file and putting some content into it.
- Save the file and view it in both **cat**, **less** and **more**.
- Go back into the file in vi and enter some more content.
- Move around the content using at least 6 different movement commands.
- Play about with several of the delete commands, especially the ones that incorporate a movement command. Remember you may undo your changes so you don't have to keep putting new content in.

Wildcards

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

What are Wildcards?

Wildcards are a set of building blocks that allow you to create a pattern defining a set of files or directories. A basic set of wild cards are:

- `*` – represents zero or more characters.
- `?` – represents a single character.
- `[]` – represents a range of characters

We first introduce the `*`. Below we list every entry beginning with a b.

Terminal

```
1.  user@bash: pwd
2.  /home/ryan/linuxtutorialwork
3.  user@bash:
4.  user@bash: ls
5.  barry.txt blah.txt bob example.png firstfile foo1 foo2
6.  foo3 frog.png secondfile thirdfile video.mpeg
7.  user@bash:
8.  user@bash: ls b*
9.  barry.txt blah.txt bob
10. user@bash:
```

Wildcards

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

More Examples

Wildcards work just the same if the path is absolute or relative.

Terminal

```
user@bash: ls /home/ryan/linuxtutorialwork/*.txt
/home/ryan/linuxtutorialwork/barry.txt /home/ryan/linuxtutorialwork/blah.txt
user@bash:
```

We now introduce the ? operator: we are looking at each file whose 2nd letter is "i".

Terminal

```
user@bash: ls ?i*
firstfile video.mpeg
user@bash:
```

Terminal

```
user@bash: ls *.???
barry.txt blah.txt example.png frog.png
user@bash:
```

Wildcards

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

More Examples

We now introduce the range [] operator.
We are looking at every file whose name either begins with "s" or "v".

```
Terminal
1. user@bash: ls [sv]*
2. secondfile video.mpeg
3. user@bash:
```

If we want every file whose name includes a digit in it we could do the following:

```
Terminal
1. user@bash: ls *[0-9]*
2. foo1 foo2 foo3
3. user@bash:
```

We may also reverse a range using the caret (^) which means look for any character which is not one of the following.

```
Terminal
1. user@bash: ls [^a-k]*
2. secondfile thirdfile video.mpeg
3. user@bash:
```

Wildcards

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Real World Examples

Find the file type of every file in a directory.

```
Terminal
1. user@bash: file /home/ryan/*
2. bin: directory
3. Documents: directory
4. frog.png: PNG image data
5. public_html: directory
6. user@bash:
```

Move all files of type either jpg or png (image files) into another directory.

```
Terminal
1. user@bash: mv public_html/*.??g public_html/images/
2. user@bash:
```

Find out the size and modification time of the .bash_history file in every users home directory.

```
Terminal
1. user@bash: ls -lh /home/*/.bash_history
2. -rw------- 1 harry users 2.7K Jan 4 07:32 /home/harry/.bash_history
3. -rw------- 1 ryan users 3.1K Jun 12 21:16 /home/ryan/.bash_history
4. user@bash:
```

Wildcards: Exercises

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

- A good directory to play with is "/etc" which is a directory containing config files for the system. As a normal user you may view the files but you can't make any changes so we can't do any harm. Do a listing of that directory to see what's there. Then pick various subsets of files and see if you can create a pattern to select only those files.
- Do a listing of "/etc" with only files that contain an extension.
- What about only a 3 letter extension?
- How about files whose name contains an uppercase letter? (hint: [[:upper:]] may be useful here)
- Can you list files whose name is 4 characters long?

Permissions

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Permissions on files and directories specify what a particular person may or may not do.

What are they?

Linux permissions dictate 3 things you may do with a file:

- `r` read – you may view the contents of the file.
- `w` write – you may change the contents of the file.
- `x` execute – you may execute or run the file if it is a program or script.

For every file or directory 3 sets of people for whom permissions mays be specified:

- `owner` – a single person who owns the file.
- `group` – every file belongs to a single group.
- `others` – everyone else who is not in the group or the owner.

View Permissions?

To view permission it suffices to type a command: `ls -l [path]`

Terminal

```
1. user@bash: ls -l /home/ryan/linuxtutorialwork/frog.png
2. -rwxr----x 1 harry users 2.7K Jan 4 07:32
   /home/ryan/linuxtutorialwork/frog.png
3. user@bash:
```

Permissions

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Change Permissions

To change permission, a command is used: `chmod [permissions] [path]`.
chmod has permissions arguments that are made up of 3 components.

- Who are we changing the permissions for? `[ugoa]` – user (owner), group, others, all.
- Are we granting/revoking the permission – indicated with either `plus(+)` or `minus(-)`.
- Which permission are we setting? – `read (r)`, `write (w)` or `execute (x)`.

Exple: Grant the execute permission to the group. Then remove the write permission for the owner.

Terminal

```
1.  user@bash: ls -l frog.png
2.  -rwxr----x 1 harry users 2.7K Jan 4 07:32 frog.png
3.  user@bash:
4.  user@bash: chmod g+x frog.png
5.  user@bash: ls -l frog.png
6.  -rwxr-x--x 1 harry users 2.7K Jan 4 07:32 frog.png
7.  user@bash:
8.  user@bash: chmod u-w frog.png
9.  user@bash: ls -l frog.png
10. -r-xr-x--x 1 harry users 2.7K Jan 4 07:32 frog.png
11. user@bash:
```

Permissions

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Change Permissions

Exple: Don't want to assign permissions individually? We can assign multiple permissions at once.

Terminal

```
1.  user@bash: ls -l frog.png
2.  -rwxr----x 1 harry users 2.7K Jan 4 07:32 frog.png
3.  user@bash:
4.  user@bash: chmod g+wx frog.png
5.  user@bash: ls -l frog.png
6.  -rwxrwx--x 1 harry users 2.7K Jan 4 07:32 frog.png
7.  user@bash:
8.  user@bash: chmod go-x frog.png
9.  user@bash: ls -l frog.png
10. -rwxrw---- 1 harry users 2.7K Jan 4 07:32 frog.png
11. user@bash:
```

Permissions

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Setting Permissions Shorthand

It is possible to connect binary numbers with 3 digits (between 000 and 111) to octal numbers (between 0 and 7) and link those to various permissions. So we will have 3 bits and 3 permissions. If we think of 1 representing on and 0 as off, then a single octal number may be used to represent a set of permissions for a set of people. Three numbers and we can specify permissions for the user, group and others.

Terminal

```
1.  user@bash: ls -l frog.png
2.  -rw-r----x 1 harry users 2.7K Jan 4 07:32 frog.png
3.  user@bash:
4.  user@bash: chmod 751 frog.png
5.  user@bash: ls -l frog.png
6.  -rwxr-x--x 1 harry users 2.7K Jan 4 07:32 frog.png
7.  user@bash:
8.  user@bash: chmod 240 frog.png
9.  user@bash: ls -l frog.png
10. --w-r----- 1 harry users 2.7K Jan 4 07:32 frog.png
11. user@bash:
```

People often remember commonly used number sequences for different types of files and find this method quite convenient. For example 755 or 750 are commonly used for scripts.

Permissions

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Permissions for Directories

The same permissions use for files may be used for directories but with a slightly different behaviour.

- `r` – you have the baility to read the contents of the directory (ie do an ls).
- `w` – you have the ability to write into the directory (ie create files and directories).
- `x` – you have the ability to enter that directory (ie cd).

Terminal

```
1.  user@bash: ls testdir
2.  file1 file2 file3
3.  user@bash:
4.  user@bash: chmod 400 testdir
5.  user@bash: ls -ld testdir
6.  dr-------- 1 ryan users 2.7K Jan 4 07:32 testdir
7.  user@bash:
8.  user@bash: cd testdir
9.  cd: testdir: Permission denied
10. user@bash: ls testdir
11. file1 file2 file3
12. user@bash:
13. user@bash: chmod 100 testdir
14. user@bash: ls -ld testdir
15. ---x------ 1 ryan users 2.7K Jan 4 07:32 testdir
16. user@bash:
17. user@bash: ls testdir
18. user@bash: cd testdir
19. user@bash: pwd
20. /home/ryan/testdir
21. ls: cannot open directory testdir/: Permission denied
22. user@bash:
```

Permissions

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Permissions for Directories

```
Terminal
1.  user@bash: ls -ld testdir
2.  --x------- 1 ryan users 2.7K Jan 4 07:32 testdir
3.  user@bash:
4.  user@bash: cd testdir
5.  user@bash:
6.  user@bash: ls
7.  ls: cannot open directory .: Permission denied
8.  user@bash:
9.  user@bash: cat samplefile.txt
10. Kyle 20
11. Stan 11
12. Kenny 37
13. user@bash:
```

The Root User

On a Linux system there are only 2 people usually who may change the permissions of a file or directory. The owner of the file or directory and the root user. The root user is a superuser who is allowed to do anything and everything on the system.

Permissions: Exercises

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

- First off, take a look at the permissions of your home directory, then have a look at the permissions of various files in there.
- Now let's go into your linuxtutorialwork directory and change the permissions of some of the files in there. Make sure you use both the shorthand and longhand form for setting permissions and that you also use a variety of absolute and relative paths. Try removing the read permission from a file then reading it. Or removing the write permission and then opening it in vi.
- Let's play with directories now. Create a directory and put some files into it. Now play about with removing various permissions from yourself on that directory and see what you can and can't do.
- Finally, have an explore around the system and see what the general permissions are for files in other system directories such as /etc and /bin

Filters

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

What are they?

- A filter, in the context of the Linux command line, is a program that accepts textual data and then transforms it in a particular way.
- Filters are a way to take raw data, either produced by another program, or stored in a file, and manipulate it to be displayed in a way more suited to what we are after.
- These filters often have various command line options that will modify their behaviour so it is always good to check out the man page for a filter to see what is available.

In the following we are going to use the following file 'mysampledata.txt' for our examples.

Terminal

```
user@bash: cat mysampledata.txt
Fred apples 20
Susy oranges 5
Mark watermellons 12
Robert pears 4
Terry oranges 9
Lisa peaches 7
Susy oranges 12
Mark grapes 39
Anne mangoes 7
Greg pineapples 3
Oliver rockmellons 2
Betty limes 14
user@bash:
```

Filters

head ⇒ `head [-number of lines to print][path]`

- Head is a program that prints the first so many lines of its input. By default it will print the first 10 lines but we may modify this with a command line argument.

Terminal

```
1.  user@bash: head mysampledata.txt
2.  Fred apples 20
3.  Susy oranges 5
4.  Mark watermellons 12
5.  Robert pears 4
6.  Terry oranges 9
7.  Lisa peaches 7
8.  Susy oranges 12
9.  Mark grapes 39
10. Anne mangoes 7
11. Greg pineapples 3
12. user@bash:
```

Terminal

```
1.  user@bash: head -4 mysampledata.txt
2.  Fred apples 20
3.  Susy oranges 5
4.  Mark watermellons 12
5.  Robert pears 4
6.  user@bash:
```

Filters

tail ⇒ `tail [-number of lines to print][path]`

- Tail is the opposite of head. Tail is a program that prints the last so many lines of its input. By default it will print the last 10 lines but we may modify this with a command line argument.

Terminal

```
1.  user@bash: tail mysampledata.txt
2.  Mark watermellons 12
3.  Robert pears 4
4.  Terry oranges 9
5.  Lisa peaches 7
6.  Susy oranges 12
7.  Mark grapes 39
8.  Anne mangoes 7
9.  Greg pineapples 3
10. Oliver rockmellons 2
11. Betty limes 14
12. user@bash:
```

Terminal

```
1.  user@bash: tail -3 mysampledata.txt
2.  Greg pineapples 3
3.  Oliver rockmellons 2
4.  Betty limes 14
5.  user@bash:
```

Filters

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

sort ⇒ `sort [-options][path]`

- Sort will sort its input, nice and simple. By default it will sort alphabetically but there are many options available to modify the sorting mechanism. Be sure to check out the man page to see everything it may do.

Terminal

```
1.  user@bash: sort mysampledata.txt
2.  Anne mangoes 7
3.  Betty limes 14
4.  Fred apples 20
5.  Greg pineapples 3
6.  Lisa peaches 7
7.  Mark grapes 39
8.  Mark watermellons 12
9.  Oliver rockmellons 2
10. Robert pears 4
11. Susy oranges 12
12. Susy oranges 5
13. Terry oranges 9
14. user@bash:
```

Filters

nl ⇒ `nl [-options] [path]`

- nl stands for number lines and it does just that.

Terminal

```
1.  user@bash: nl mysampledata.txt
2.  1 Fred apples 20
3.  2 Susy oranges 5
4.  3 Mark watermellons 12
5.  4 Robert pears 4
6.  5 Terry oranges 9
7.  6 Lisa peaches 7
8.  7 Susy oranges 12
9.  8 Mark grapes 39
10. 9 Anne mangoes 7
11. 10 Greg pineapples 3
12. 11 Oliver rockmellons 2
13. 12 Betty limes 14
14. user@bash:
```

Filters

nl ⇒ `nl [-options] [path]`

- The basic formatting is ok but sometimes you are after something a little different. With a few command line options, nl is happy to oblige.

Terminal

```
user@bash: nl -s '. ' -w 10 mysampledata.txt
         1. Fred apples 20
         2. Susy oranges 5
         3. Mark watermellons 12
         4. Robert pears 4
         5. Terry oranges 9
         6. Lisa peaches 7
         7. Susy oranges 12
         8. Mark grapes 39
         9. Anne mangoes 7
        10. Greg pineapples 3
        11. Oliver rockmellons 2
        12. Betty limes 14
user@bash:
```

Filters

wc ⇒ `wc [-options][path]`

- wc stands for word count and it does just that (as well as characters and lines). By default it will give a count of all 3 but using command line options we may limit it to just what we are after. Options `-l` (lines), `-m` (characters), `-w` (words) are possible.

Terminal

```
1. user@bash: wc mysampledata.txt
2. 12 36 195 mysampledata.txt
3. user@bash:
```

Terminal

```
1. user@bash: wc -l mysampledata.txt
2. 12 mysampledata.txt
3. user@bash:
```

Terminal

```
1. user@bash: wc -lw mysampledata.txt
2. 12 36 mysampledata.txt
3. user@bash:
```

Filters

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

cut ⇒ `cut [-options][path]`

- cut is a nice little program to use if your content is separated into fields (columns) and you only want certain fields.
- In our sample file we have our data in 3 columns, the first is a name, the second is a fruit and the third an amount. Let's say we only wanted the first column.

Terminal

```
user@bash: cut -f 1 -d ' ' mysampledata.txt
Fred
Susy
Mark
Robert
Terry
Lisa
Susy
Mark
Anne
Greg
Oliver
Betty
user@bash:
```

Filters

cut ⇒ `cut [-options][path]`

- cut defaults to using the TAB character as a separator to identify fields. In our file we have used a single space instead so we need to tell cut to use that instead.
- The separator character may be anything you like, for instance in a CSV file the separator is typically a comma (,). This is what the `-d` option does (we include the space within single quotes so it knows this is part of the argument). The `-f` option allows us to specify which field or fields we would like. If we wanted 2 or more fields then we separate them with a comma as below.

Terminal

```
1.  user@bash: cut -f 1,2 -d ' ' mysampledata.txt
2.  Fred apples
3.  Susy oranges
4.  Mark watermellons
5.  Robert pears
6.  Terry oranges
7.  Lisa peaches
8.  Susy oranges
9.  Mark grapes
10. Anne mangoes
11. Greg pineapples
12. Oliver rockmellons
13. Betty limes
14. user@bash:
```

Filters

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

sed ⇒ `sed <expression> [path]`

- sed stands for Stream Editor and it effectively allows us to do a search and replace on our data. It is quite a powerful command but we will use it here in it's basic format.
- A basic expression is of the following format: `s/search/replace/g`.
- The initial `s` stands for substitute and specifies the action to perform. Then between the first and second slashes (/) we place what it is we are searching for. Then between the second and third slashes, what it is we wish to replace it with. The `g` at the end stands for `global` and is optional. If we omit it then it will only replace the first instance of search on each line. With the g option we will replace every instance of search that is on each line.

Terminal

```
1.  user@bash: sed 's/oranges/bananas/g' mysampledata.txt
2.  Fred apples 20
3.  Susy bananas 5
4.  Mark watermellons 12
5.  Robert pears 4
6.  Terry bananas 9
7.  Lisa peaches 7
8.  Susy bananas 12
9.  Mark grapes 39
10. Anne mangoes 7
11. Greg pineapples 3
12. Oliver rockmellons 2
13. Betty limes 14
14. user@bash:
```

Filters

uniq ⇒ `uniq [options] [path]`

- uniq stands for unique and it's job is to remove duplicate lines from the data. One limitation however is that those lines must be adjacent (ie, one after the other).
- Let's say that our sample file was actually generated from another sales program but after a software update it had some buggy output.

Terminal

```
1.  user@bash: cat mysampledata.txt
2.  Fred apples 20
3.  Susy oranges 5
4.  Susy oranges 5
5.  Susy oranges 5
6.  Mark watermellons 12
7.  Robert pears 4
8.  Terry oranges 9
9.  Lisa peaches 7
10. Susy oranges 12
11. Mark grapes 39
12. Mark grapes 39
13. Anne mangoes 7
14. Greg pineapples 3
15. Oliver rockmellons 2
16. Betty limes 14
17. user@bash:
```

Filters

uniq ⇒ `uniq [options] [path]`

- uniq stands for unique and it's job is to remove duplicate lines from the data. One limitation however is that those lines must be adjacent (ie, one after the other).
- No worries, we can easily fix that using uniq.

Terminal

```
1.  user@bash: uniq mysampledata.txt
2.  Fred apples 20
3.  Susy oranges 5
4.  Mark watermellons 12
5.  Robert pears 4
6.  Terry oranges 9
7.  Lisa peaches 7
8.  Susy oranges 12
9.  Mark grapes 39
10. Anne mangoes 7
11. Greg pineapples 3
12. Oliver rockmellons 2
13. Betty limes 14
14. user@bash:
```

Filters

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

tac ⇒ `tac [path]`

- The program tac is actually cat in reverse. It was named this as it does the opposite of cat. Given data it will print the last line first, through to the first line.
- Maybe our sample file is generated by writing each new order to the end of the file. As a result, the most recent orders are at the end of the file. We would like it the other way so that the most recent orders are always at the top.

Terminal

```
user@bash: tac mysampledata.txt
Betty limes 14
Oliver rockmellons 2
Greg pineapples 3
Anne mangoes 7
Mark grapes 39
Susy oranges 12
Lisa peaches 7
Terry oranges 9
Robert pears 4
Mark watermellons 12
Susy oranges 5
Fred apples 20
user@bash:
```

Filters: Exercises

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

- First off, you may want to make a file with data similar to our sample file.
- Now play with each of the programs we looked at above. Make sure you use both relative and absolute paths.
- Have a look at the man page for each of the programs and try at least 2 of the command line options for them.

Grep and Regular Expressions

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

What are they?

- Regular expressions are similar to the wildcards as they allow us to create a pattern.
- Regular expressions are typically used to identify and manipulate specific pieces of data. EG. We may wish to identify every line which contains an email address or a url in a set of data
- We will use a similar sample file as before, included below as a reference.

Terminal

```
1.  user@bash: cat mysampledata.txt
2.  Fred apples 20
3.  Susy oranges 5
4.  Mark watermellons 12
5.  Robert pears 4
6.  Terry oranges 9
7.  Lisa peaches 7
8.  Susy oranges 12
9.  Mark grapes 39
10. Anne mangoes 7
11. Greg pineapples 3
12. Oliver rockmellons 2
13. Betty limes 14
14. user@bash:
```

Grep and Regular Expressions

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

eGrep ⇒ `egrep [comman line options]<pattern>[path]`

- egrep is a program which will search a given set of data and print every line which contains a given pattern.
- It is an extension of a program called grep. Its name is odd but based upon a command which did a similar function, in a text editor called ed. It has many command line options which modify its behaviour so its worth checking its man page.
- The -v option tells grep to instead print every line which does not match the pattern.
- If we want to identify every line which contains the string 'mellon'

Terminal

```
1. user@bash: egrep 'mellon' mysampledata.txt
2. Mark watermellons 12
3. Oliver rockmellons 2
4. user@bash:
```

Grep and Regular Expressions

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

eGrep ⇒ `egrep [comman line options]<pattern>[path]`

- Sometimes we want to know not only which lines matched but their line number as well.

Terminal

```
1. user@bash: egrep -n 'mellon' mysampledata.txt
2. 3:Mark watermellons 12
3. 11:Oliver rockmellons 2
4. user@bash:
```

- Maybe we are not interested in seeing the matched lines but wish to know how many lines did match.

Terminal

```
1. user@bash: egrep -c 'mellon' mysampledata.txt
2. 2
3. user@bash:
```

Grep and Regular Expressions

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Regular Expressions Overview

- `.(dot)` – a single character.
- `?` – the preceding character matches 0 or 1 times only.
- `*` – the preceding character matches 0 or more times.
- `+` – the preceding character matches 1 or more times.
- `n` – the preceding character matches exactly n times.
- `n,m` – the preceding character matches at least n times and not more than m times.
- `[agd]` – the character is one of those included within the square brackets.
- `[^agd]` – the character is not one of those included within the square brackets.
- `[c-f]` – the dash within the square brackets operates as a range. In this case it means either the letters c, d, e or f.
- `()` – allows us to group several characters to behave as one.
- `|(pipe symbol)` – the logical OR operation.
- `^` – matches the beginning of the line.
- `$` – matches the end of the line.

Grep and Regular Expressions

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Some Examples

- Let's say we wish to identify any line with two or more vowels in a row. In the example below the multiplier **2,** applies to the preceding item which is the range.

```
Terminal
1.  user@bash: egrep '[aeiou]{2,}' mysampledata.txt
2.  Robert pears 4
3.  Lisa peaches 7
4.  Anne mangoes 7
5.  Greg pineapples 3
6.  user@bash:
```

- How about any line with a 2 on it which is not the end of the line. In this example the multiplier + applies to the . which is any character.

```
Terminal
1.  user@bash: egrep '2.+' mysampledata.txt
2.  Fred apples 20
3.  user@bash:
```

Grep and Regular Expressions

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Some Examples

- The number 2 as the last character on the line.

```
Terminal
1. user@bash: egrep '2$' mysampledata.txt
2. Mark watermellons 12
3. Susy oranges 12
4. Oliver rockmellons 2
5. user@bash:
```

- And now each line which contains either 'is' or 'go' or 'or'.

```
Terminal
1. user@bash: egrep 'or|is|go' mysampledata.txt
2. Susy oranges 5
3. Terry oranges 9
4. Lisa peaches 7
5. Susy oranges 12
6. Anne mangoes 7
7. user@bash:
```

- Maybe we wish to see orders for everyone who's name begins with A - K.

```
Terminal
1. user@bash: egrep '^[A-K]' mysampledata.txt
2. Fred apples 20
3. Anne mangoes 7
4. Greg pineapples 3
5. Betty limes 14
6. user@bash:
```

Grep and Regular Expressions: Exercises

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

- You may want to make a file with data similar to our sample file.
- Now play with some of the examples we looked at above.
- Have a look at the man page for egrep and try atleast 2 of the command line options for them.

Piping and Redirection

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

What are They?

- Every program we run on the command line automatically has three data streams connected to it.
- STDIN (0) – Standard input (data fed into the program)
- STDOUT (1) – Standard output (data printed by the program, defaults to the terminal)
- STDERR (2) – Standard error (for error messages, also defaults to the terminal)

- Every program we run on the command line automatically has three data streams connected to it.
- Piping and redirection is the means by which we may connect these streams between programs and files to direct data in interesting and useful ways.

Piping and Redirection

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Redirecting to a File

- We normally get our output on the screen which is convenient usually.
- We sometimes may wish to save it into a file to keep as a record for example.
- The greater than operator (>) indicates to the command line that we wish the programs output (or whatever it sends to STDOUT) to be saved in a file instead of printed to the screen.

Terminal

```
1.  user@bash: ls
2.  barry.txt bob example.png firstfile foo1 video.mpeg
3.  user@bash: ls > myoutput
4.  user@bash: ls
5.  barry.txt bob example.png firstfile foo1 myoutput video.mpeg
6.  user@bash: cat myoutput
7.  barry.txt
8.  bob
9.  example.png
10. firstfile
11. foo1
12. myoutput
13. video.mpeg
14. user@bash:
```

Piping and Redirection

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Saving to an Existing File

- If we redirect to a file which does not exist, it will be created automatically for us. If we save into a file which already exists, however, then its contents will be cleared, then the new output saved to it.

Terminal

```
1.  user@bash: cat myoutput
2.  barry.txt
3.  bob
4.  example.png
5.  firstfile
6.  foo1
7.  myoutput
8.  video.mpeg
9.  user@bash: wc -l barry.txt > myoutput
10. user@bash: cat myoutput
11. 7 barry.txt
12. user@bash:
```

Piping and Redirection

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Saving to an Existing File

- We can instead get the new data to be appended to the file by using the double greater than operator (>>).

Terminal

```
user@bash: cat myoutput
7 barry.txt
user@bash: ls >> myoutput
user@bash: cat myoutput
7 barry.txt
barry.txt
bob
example.png
firstfile
foo1
myoutput
video.mpeg
user@bash:
```

Piping and Redirection

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Redirecting from a File

- If we use the less than operator (<) then we can send data the other way. We will read data from the file and feed it into the program via its STDIN stream.

Terminal

```
1. user@bash: wc -l myoutput
2. 8 myoutput
3. user@bash: wc -l < myoutput
4. 8
5. user@bash:
```

- We may easily combine the two forms of redirection we have seen so far into a single command.

Terminal

```
1. user@bash: wc -l < barry.txt > myoutput
2. user@bash: cat myoutput
3. 7
4. user@bash:
```

Piping and Redirection

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Redirecting STDERR

- The three streams actually have numbers associated with them. STDERR is stream number 2 and we may use these numbers to identify the streams.
- If we place a number before the ¿ operator then it will redirect that stream (if we don't use a number, like we have been doing so far, then it defaults to stream 1).

Terminal

```
1.  user@bash: ls -l video.mpg blah.foo
2.  ls: cannot access blah.foo: No such file or directory
3.  -rwxr--r-- 1 ryan users 6 May 16 09:14 video.mpg
4.  user@bash: ls -l video.mpg blah.foo 2> errors.txt
5.  -rwxr--r-- 1 ryan users 6 May 16 09:14 video.mpg
6.  user@bash: cat errors.txt
7.  ls: cannot access blah.foo: No such file or directory
8.  user@bash:
```

Piping and Redirection

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Redirecting STDERR

- We may wish to save both normal output and error messages into a single file. This can be done by redirecting the STDERR stream to the STDOUT stream and redirecting STDOUT to a file. We redirect to a file first then redirect the error stream. We identify the redirection to a stream by placing an & in front of the stream number (otherwise it would redirect to a file called 1).

Terminal

```
1. user@bash: ls -l video.mpg blah.foo > myoutput 2>&1
2. user@bash: cat myoutput
3. ls: cannot access blah.foo: No such file or directory
4. -rwxr--r-- 1 ryan users 6 May 16 09:14 video.mpg
5. user@bash:
```

Piping and Redirection

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Piping

- The mechanism for sending data from one program to another is called piping and the operator we use is (|)
- This operator feeds the output from the program on the left as input to the program on the right.

Terminal

```
1. user@bash: ls
2. barry.txt bob example.png firstfile foo1 myoutput video.mpeg
3. user@bash: ls | head -3
4. barry.txt
5. bob
6. example.png
7. user@bash:
```

Piping and Redirection

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Piping

- We may pipe as many programs together as we like.
- Below we have piped the output to **tail** so as to get only the third file.

Terminal

```
1. user@bash: ls | head -3 | tail -1
2. example.png
3. user@bash:
```

- We may combine pipes and redirection too.

Terminal

```
1. user@bash: ls | head -3 | tail -1 > myoutput
2. user@bash: cat myoutput
3. example.png
4. user@bash:
```

Piping and Redirection

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Examples

- We sort the listing of a directory so that all the directories are listed first.

Terminal

```
1. user@bash: ls -l /etc | tail -n +2 | sort
2. drwxrwxr-x 3 nagios nagcmd 4096 Mar 29 08:52 nagios
3. drwxr-x--- 2 news news 4096 Jan 27 02:22 news
4. drwxr-x--- 2 root mysql 4096 Mar 6 22:39 mysql
5. ...
6. user@bash:
```

- We feed the output of a program into the program less so that we can view it easier.

Terminal

```
1. user@bash: ls -l /etc | less
2. (Full screen of output you may scroll. Try it yourself to see.)
```

Piping and Redirection

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Examples

- Identify all files in your home directory which the group has write permission for.

```
Terminal
1.  user@bash: ls -l ~ | grep '^.....w'
2.  drwxrwxr-x 3 ryan users 4096 Jan 21 04:12 dropbox
3.  user@bash:
```

- Create a listing of every user which owns a file in a given directory as well as how many files and directories they own.

```
Terminal
1.  user@bash: ls -l /projects/ghosttrail | tail -n +2 | sed 's/\s\s*/
    /g' | cut -d ' ' -f 3 | sort | uniq -c
2.  8 anne
3.  34 harry
4.  37 tina
5.  18 ryan
6.  user@bash:
```

Piping and Redirection: Exercises

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

- Experiment with saving output from various commands to a file. Overwrite the file and append to it as well. Make sure you are using a both absolute and relative paths as you go.
- Now see if you can list only the 20th last file in the directory /etc.
- Finally, see if you can get a count of how many files and directories you have the execute permission for in your home directory.

Process Management

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

What Are They?

- A program is a series of instructions that tell the computer what to do.
- When we run a program, those instructions are copied into memory and space is allocated for variables and other stuff required to manage its execution.
- This running instance of a program is called a process and it's processes which we manage.

Process Management

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

What is Currently Running? ⇒ `top`

- Linux is a multitasking operating system.
- This means that many processes can be running at the same time.
- As well as the processes we are running, there may be other users on the system also running stuff and the OS itself will usually also be running various processes which it uses to manage everything in general. If we would like to get a snapshot of what is currently happening on the system we may use a program called top.
- An alternative approach is using ps which stands for processes show the processes running in your current terminal (`ps [aux]`). With the additional argument aux it shows a complete system view.

Terminal

```
1.  user@bash: top
2.  Tasks: 174 total, 3 running, 171 sleeping, 0 stopped
3.  KiB Mem: 4050604 total, 3114428 used, 936176 free
4.  Kib Swap: 2104476 total, 18132 used, 2086344 free
5.
6.   PID USER %CPU %MEM COMMAND
7.  6978 ryan 3.0  21.2 firefox
8.    11 root 0.3   0.0 rcu_preempt
9.  6601 ryan 2.0   2.4 kwin
10. ...
```

Process Management

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Killing a Crashed Process? ⇒ `kill [signal] <PID>`

- Let's suppose you have process not working ('Firefox' say). It's possible to identify it, kill it and reopen it later.

Terminal

```
1. user@bash: ps aux | grep 'firefox'
2. ryan 6978 8.8 23.5 2344096 945452 ? Sl 08:03 49:53 /usr/lib64/firefox/firefox
3. user@bash:
```

Terminal

```
1. user@bash: kill 6978
2. user@bash: ps aux | grep 'firefox'
3. ryan 6978 8.8 23.5 2344096 945452 ? Sl 08:03 49:53 /usr/lib64/firefox/firefox
4. user@bash:
```

Terminal

```
1. user@bash: kill -9 6978
2. user@bash: ps aux | grep 'firefox'
3. user@bash:
```

Process Management

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Foreground and Background Jobs ⇒ `jobs`

- When we run a command by default, it is run on the foreground. If you append the ampersand (&) at the end of the command then we are telling the terminal to run this process in the background.
- Applying the & you will notice that it assigns the process a job number an tells us what that number is, and gives us the prompt back straight away. We can continue working while the process runs in the background

Terminal

```
1. user@bash: sleep 5
2. user@bash:
```

Terminal

```
1. user@bash: sleep 5 &
2. [1] 21634
3. user@bash:
4. user@bash:
5. [1]+ Done sleep 5
```

Process Management

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Foreground and Background Jobs ⇒ `fg <job number>`

- We can move jobs between the foreground and background as well.
- If you press `CTRL + z` then the currently running foreground process will be paused and moved into the background.
- We can then use a program called `fg` which stands for foreground to bring background processes into the foreground.

Terminal

```
1.  user@bash: sleep 15 &
2.  [1] 21637
3.  user@bash: sleep 10
4.  (you press CTRL + z, notice the prompt comes back.)
5.  user@bash: jobs
6.  [1]- Running sleep 15 &
7.  [2]+ Stopped sleep 10
8.  user@bash: fg 2
9.  [1] Done sleep 15
10. user@bash:
```

Process Management: Exercises

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

- Start a few programs in your desktop. Then use ps to identify their PID and kill them.

- Play about with the command sleep and moving processes between the foreground and background.

Bash Scripting

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

What are They?

- A bash script is a document or file stating what to say and what to do by the computer.
- A bash script allows us to define a series of actions which the computer will then perform without us having to enter the commands ourselves. If a particular task is done often, or it is repetitive, then a script can be a useful tool.
- A Bash script is interpreted (read and acted upon) by something called an interpreter.
- Anything you can run on the command line you may place into a script and they will behave exactly the same. Vice versa, anything you can put into a script, you may run on the command line and again it will perform exactly the same.
- A script is just a plain text file and it may have any name you like. You create them the same way you would any other text file, with just a plain old text editor.

Bash Scripting

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

An Example ⇒ `echo <message>`

- This script will print a message to the screen (using a program called echo) then give us a listing of what is in our current directory.

Terminal

```
1.  user@bash: cat myscript.sh
2.  #!/bin/bash
3.  # A simple demonstration script
4.  # Ryan 13/6/2019
5.
6.  echo Here are the files in your current directory:
7.  ls
8.  user@bash:
9.  user@bash: ls -l myscript.sh
10. -rwxr-xr-x 1 ryan users 2 Jun 4 2012 myscript.sh
11. user@bash:
12. user@bash: ./myscript.sh
13. Here are the files in your current directory:
14. barry.txt bob example.png firstfile foo1 myoutput video.mpeg
15. user@bash:
```

Bash Scripting

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

The Shebang ⇒ `which <program>`

- The very first line of a script should tell the system which interpreter should be used on this file. It is important that this is the very first line of the script. It is also important that there are no spaces. The first two characters `#!` (the shebang) tell the system that directly after it will be a path to the interpreter to be used. If we don't know where our interpreter is located then we may use a program called which to find out.

Terminal

```
1. user@bash: which bash
2. /bin/bash
3. user@bash:
4. user@bash: which ls
5. /usr/bin/ls
6. user@bash:
```

The Name

- Linux is an extensionless system. That means we may call our script whatever we like and it will not affect it's running in any way. While it is typical to put a `.sh` extension on our scripts, this is purely for convenience and is not required. We could name our script above simply **myscript** or even **myscript.jpg** and it would still run quite happily.

Bash Scripting

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Comment

- A comment is just a note in the script that does not get run, it is merely there for your benefit.
- Comments are easy to put in, all you need to do is place a hash (#) then anything after that is considered a comment.
- A comment can be a whole line or at the end of a line..

Terminal

```
1. user@bash: cat myscript.sh
2. #!/bin/bash
3. # A comment which takes up a whole line
4. ls # A comment at the end of the line
5. user@bash:
```

- It is common practice to include a comment at the top of a script with a brief description of what the script does and also who wrote it and when.
- For the rest of the script, it is not necessary to comment every line. It will be self explanatory what most lines they do. Only put comments in for important lines or to explain a particular command whose operation may not be immediately obvious.

Bash Scripting

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

The "./":

- When we type a command on the command line, the system runs through a preset series of directories, looking for the program we specified.
- We may find out these directories by looking at a particular variable `PATH`.

Terminal

```
1. user@bash: echo $PATH
2. /usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/X11R6/bin:/usr/games:/usr/lib/mit/bin:/usr/lib/m
3. user@bash:
```

- The system will look in the first directory and if it finds the program it will run it, if not it will check the second directory and so on. Directories are separated by a colon (:).
- The system will not look in any directories (not even your current directory) apart from these. We can override this behaviour however by supplying a path: this will allow the system to ignore the PATH and go straight to the location you have specified.

Permissions

- A script must have the execute permission before we may run it (even if we are the owner of the file). For safety reasons, you don't have execute permission by default so you have to add it. A good command to run to ensure your script is set up right is `chmod 755 <script>`.

Bash Scripting

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Variables

- A variable is a container for a simple piece of data.
- When we set a variable, we specify it's name, followed directly by an equal sign (=) followed directly by the value. (So, no spaces on either side of the = sign.)
- When we refer to a variable, we must place a dollar sign ($) before the variable name.

Terminal

```
1.  user@bash: cat variableexample.sh
2.  #!/bin/bash
3.  # A simple demonstration of variables
4.  # Ryan 13/6/2019
5.
6.  name='Ryan'
7.  echo Hello $name
8.  user@bash:
9.  user@bash: ./variableexample.sh
10. Hello Ryan
11. user@bash:
```

Bash Scripting

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Variables: Command Line Argument

When we run a script, there are several variables that get set automatically for us.

- $0 – The name of the script.
- $1--$9 – Any command line arguments given to the script. $1 is the first argument, $2 the second and so on.
- $# – How many command line arguments were given to the script.
- $* – All of the command line arguments.

Terminal

```
1.  user@bash: cat morevariables.sh
2.  #!/bin/bash
3.  # A simple demonstration of variables
4.  # Ryan 13/6/2019
5.
6.  echo My name is $0 and I have been given $# command line arguments
7.  echo Here they are: $*
8.  echo And the 2nd command line argument is $2
9.  user@bash:
10. user@bash: ./morevariables.sh bob fred sally
11. My name is morevariables.sh and I have been given 3 command line arguments
12. Here they are: bob fred sally
13. And the 2nd command line argument is fred
14. user@bash:
```

Bash Scripting

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Variables: Back Ticks

You can save the output of a command to a variable and the mechanism we use for that is the `backtick (‘)`.

Terminal

```
1.  user@bash: cat backticks.sh
2.  #!/bin/bash
3.  # A simple demonstration of using backticks
4.  # Ryan 13/6/2019
5.
6.  lines=`cat $1 | wc -l`
7.  echo The number of lines in the file $1 is $lines
8.  user@bash:
9.  user@bash: ./backticks.sh testfile.txt
10. The number of lines in the file testfile.txt is 12
11. user@bash:
```

Bash Scripting

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

Variables: Backup Script

A sample backup script that you could organize for your various research projects.

Terminal

```
user@bash: cat projectbackup.sh
#!/bin/bash
# Backs up a single project directory
# Ryan 13/6/2019

date=`date +%F`
mkdir ~/projectbackups/$1_$date
cp -R ~/projects/$1 ~/projectbackups/$1_$date
echo Backup of $1 completed
user@bash:
user@bash: ./projectbackup.sh ocelot
Backup of ocelot completed
user@bash:
```

Bash Scripting

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

If Statements

A sample script which exhibits the use of the if statement.

Terminal

```
user@bash: cat projectbackup.sh
#!/bin/bash
# Backs up a single project directory
# Ryan 13/6/2019

if [ $# != 1 ]
then
    echo Usage: A single argument which is the directory to backup
    exit
fi
if [ ! -d ~/projects/$1 ]
then
    echo 'The given directory does not seem to exist (possible typo?)'
    exit
fi
date=`date +%F`

# Do we already have a backup folder for todays date?
if [ -d ~/projectbackups/$1_$date ]
then
    echo 'This project has already been backed up today, overwrite?'
    read answer
    if [ $answer != 'y' ]
    then
        exit
    fi
else
    mkdir ~/projectbackups/$1_$date
fi
cp -R ~/projects/$1 ~/projectbackups/$1_$date
echo Backup of $1 completed
user@bash:
```

Bash Scripting: Exercises

United Nations Educational, Scientific and Cultural Organization

ICTP - East African Institute for Fundamental Research under the auspices of UNESCO

- Think about writing your own backup script. You can make it as simple or complex as you like. Maybe start off with a really simple one and progressively improve it.
- See if you can write a script that will give you a report about a given directory. Things you could report on include:
  - How many files are in the directory?
  - How many directories are in the directory?
  - What is the biggest file?
  - What is the most recently modified or created file?
  - A list of people who own files in the directory.
  - Anything else you can think of.