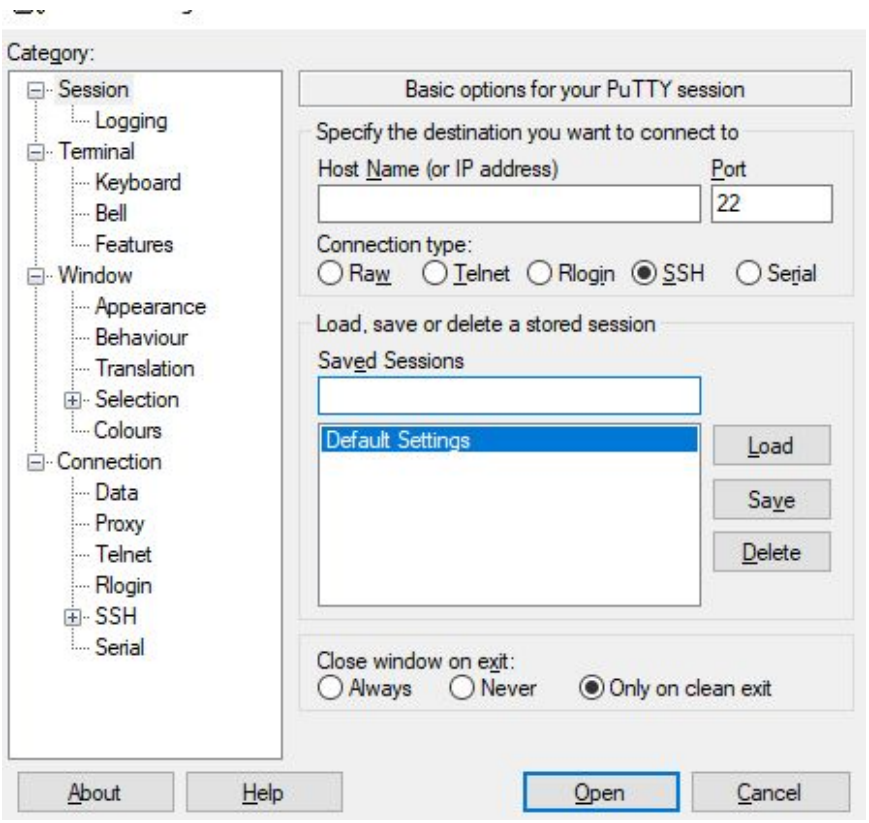
Bash tutorial share-doc

1. Intro (Marika)

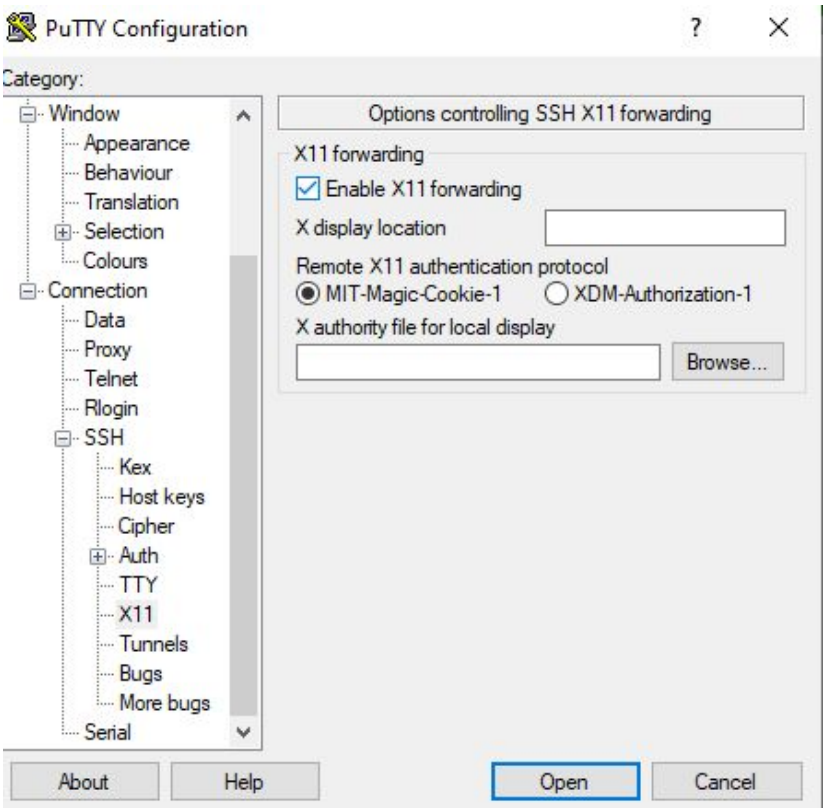
Access to the linux machine

When you have your account on the linux machine in the office you will be able to use it in your own pc with some basic steps.

- Download and install PuTTY and Xming from http://www.geo.mtu.edu/geoschem/docs/putty_install.html
- open Xming;
- open PuTTY, you will see something like this:



- Go to SSH > X11 on the left menu and tick “X11 forwarding”;



- Go to session and insert your Host Name, that will be in the format username@pompeii.bg.ic.ac.uk and press Open. Insert your password.

Now you are logged in and you can use the linux machine in Neal's office by the terminal. To log out just digit `exit` on the terminal.

You must be connected to the Imperial network to have access to that machine, but you can also configure an Imperial VPN connection to use it also when you are using a different network. The instructions for the configuration are reported in the Imperial website: https://www.imperial.ac.uk/admin-services/ict/self-service/connect-communicate/remote-access/virtual-private-network-vpn/.

Basic navigation

You can start becoming comfortable in linux environment using some basic commands.

When you first login, your current working directory is your home directory. To use a command on a file (or directory) not in the current working directory (the directory you are currently in), you must either cd to the correct directory, or specify its full pathname.

Home directories can also be referred to by the tilde ~ character. It can be used to specify paths starting at your home directory.

pwd: Print Working Directory

`pwd` is used to print the current working directory.

clear

Use clear to clean the terminal window.

ls

ls can be used with no arguments and it will simply list all of the files in the current working directory. If one or more arguments are specified it will list the files in the directories passed as arguments.

ls does not cause all the files in your home directory to be listed, but only those ones whose name does not begin with a dot (.). Files beginning with a dot (.) are known as hidden files and usually contain important program configuration information. To list all files in your home directory including those whose names begin with a dot, type:

```
ls -a
```

It is an example of a command which can take options: -a is an example of an option. The options change the behaviour of the command.

mkdir: make directory

With this command you can create new directories. They will be created in the current one.

Example:

```
$ mkdir example_dir
```

```
$ ls
```

The output will be:

```
username@pompeii:~/$
```

```
example_dir
```

cd: Change Directory

Without parameters this command will move you on your home directory. You can specify another directory to move with `cd directoryname`. You can specify relative directories (just start typing in the directory that is in the current location) or an absolute directory (start with a / and type everything from there).

Example:

```
$ cd example_dir

$ pwd

username@pompeii:~/example_dir$
```

You can also use `cd ..`, it will move you one directory up the hierarchy (back to your home directory in this case).

mv: move

Move a file or a directory to another location. You can give two file names and it will rename the first file to the name of the second, or you can specify one or more files followed by a final directory, in which case it will move all of the files into the last directory.

Example:

```
$ mkdir test

$ mv test example_dir

$ cd example_dir

$ ls$

test
```

It can also be used to rename a file, by moving the file to the same directory, but giving it a different name.

Example:

```
$ls

file1.txt

$ mv file.txt file2.txt

$ls

file2.txt
```

cp: copy

Copy a file. This command has to be run with 2 parameters: the file to be copied and the name of the copied file.

Example:

```
$ cp file1.txt file2.txt
```

This command copies the content of *file1.txt* and save it as *file2.txt*.

```
$ ls

test file1.txt file2.txt
```

rm and rmdir

`rm` is used to remove a file, it will delete files given as parameters. `rmdir` has the same function but for directories. It will only remove EMPTY directories.

Example:

```
$ rm file1.txt

$ ls

test file2.txt

$rmdir test

$ls

file2.txt
```

Exit

Terminate the current session.

Terminal tips

Some Tips about using the terminal:

- If you type “man” followed by the name of a command , it will pull up a screen of information about that command. You can press ’?’ to see help on how to navigate, ’j’ to move down, ’k’ to move up, and ’q’ to quit.
- tab completion: when you are typing a command in the command line if you hit the Tab button, the terminal will attempt to complete what you were typing. This is typically used when typing in file and directory names.
- History: if you want to repeat a command that you have previously used, simply press the up arrow. If you type ’!’ followed by any other characters, the terminal will execute the most recent command that started with the characters that you typed in.

Summary

pwd	print current directory
clear	Clear the terminal window
ls	list files in the current directory
mkdir	create a directory
cd	change directory
mv	move a file/directory, used also to rename
cp	copy a file
rm	remove a file
rmdir	remove a directory
exit	termite the current session

Text editors

Everything in UNIX is either a file or a process. A process is an executing program identified by a unique PID (process identifier). A file is a collection of data. They are created by users using text editors, running compilers etc.

The first thing that you can learn to do is to use a simple text editor.

Nano

Nano is a basic text editor for linux, it is very simple and you can use it for writing bash scripts or tweak configuration files. It can be opened from the terminal simply writing “nano”. If you want to edit an existing file just type `nano filename`. If you want to open a file with the cursor on a specific line and character use the following syntax:

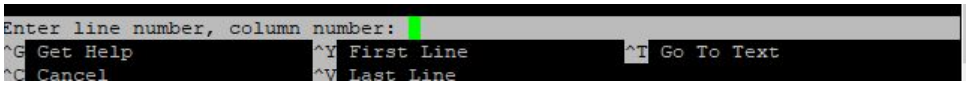
```
nano +line_number,character_number filename
```

The interface is shown in figure below.

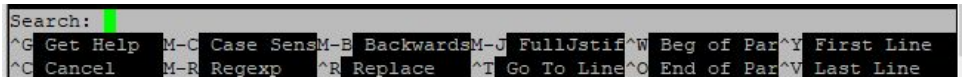
At the bottom of the window, there is a list of the most basic command shortcuts to use with the nano editor, the symbol (^) represents the Ctrl key.

You can start typing and editing the text immediately after opening the file. If you want to save just press [ctrl] + [o]. If you hav never saved the file before nano will ask you the name for the text file, press [enter]. To exit: [ctrl]+[x]. Some other useful commands are:

- [Ctrl]+[c] – print out the current line number;
- [Ctrl]+[r] – read in from another file and insert in the current file;
- [Ctrl]+[g] help file.
- [Ctrl]+[_] - moves the cursor to a specific line and character number. The menu on the bottom of the screen will change. Enter the number(s) in the “Enter line number, column number:” field and hit Enter



- [Ctrl]+[w] - to search. The menu will change again asking you to insert the term to search.



  To move to the next match, press Alt+w.

- [Ctrl]+[\] - search and replace. Enter the search term and the text to be replaced with. The editor will move to the first match and ask you whether to replace it. After hitting Y or N it will move to the next match. Pressing A will replace all matches.
- [Alt]+[a] - select text. Move the cursor to the end of the text you want to select using the arrow keys. If you want to:
  - Ctrl+6 - to cancel the selection

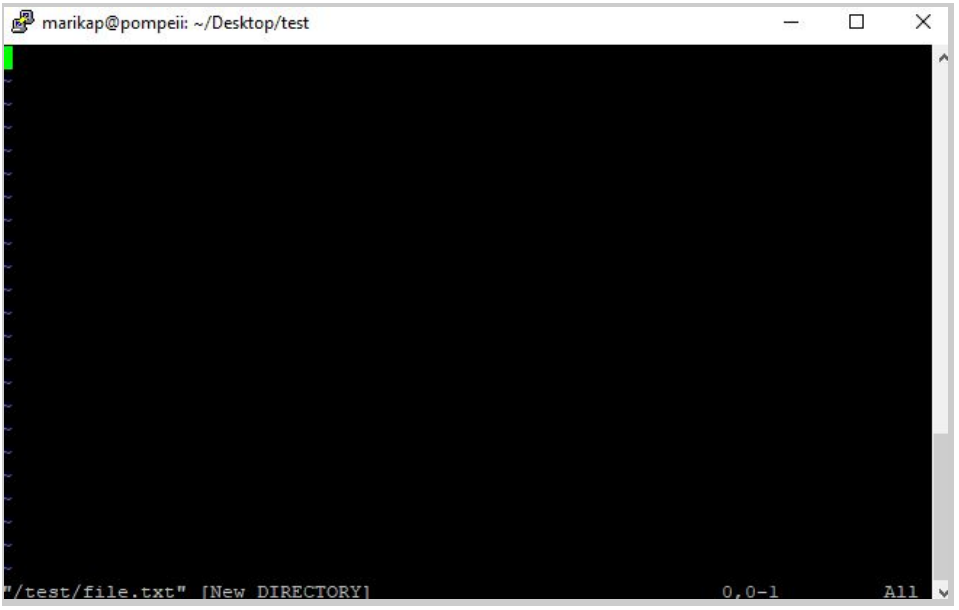
        - Alt+6 - to copy the selected text to the clipboard use the command.
        - Ctrl+k - to cut the selected text.
    - [Ctrl]+[k] – cut a line of text (without selection)
    - [Ctrl]+[u] – past the text.

Vim

Vim is an other popular text editor for Unix, it is more complicated but more powerful than nano. To start using vim, just run the "vim" command on the Linux shell. If you want to edit an existing file type “vim” followed by the path of the file that you want to edit.

Example, editing of the file /test/file.txt:

```
$vim /test/file.txt
```



There are some *modes* in which Vim can be used, now it is in command mode, this means that all the alphanumeric keys are bound to commands, rather than inserting those characters and they are used to move around the document or to make some corrections as deleting some lines.

To start editing the file content, enter `:i` and press enter. Now you are in insert mode, the word --insert-- will appear at the bottom of the editor window. You can type normally until you want to make a correction, save the file, or perform another operation. To do these things you have to switch again to command mode.

- :w to write the file. That will write the file to the existing filename. If you don’t have a filename or want to write out to a different filename, use :w *filename*.
- :help to have a list of commands
- :q to exit the editor, :q! To exit without save
- You can also exit Vim using ZZ, which will save and quit the file.

Other useful commands are:

- h moves the cursor one character to the left.
- j moves the cursor down one line.
- k moves the cursor up one line.
- l moves the cursor one character to the right.
- 0 moves the cursor to the beginning of the line.
- $ moves the cursor to the end of the line.
- w move forward one word.
- b move backward one word.
- G move to the end of the file.
- gg move to the beginning of the file.
- d starts the delete operation.
- u will undo the last operation.
- Ctrl-r will redo the last undo.
- If you want to search through the document from command mode, use / followed by the text you want to search for. If I want to find it again, I hit n, if you want the previous instance of that word use N

If you want to move up six lines, enter 6k and Vim will move the cursor up six lines and the same of other commands. You can use commands in combinations, for example dw will delete a word, d0 will delete to the beginning of a line, d$ will delete to the end of a line, dgg will delete to the beginning of the file, dG will delete to the end of the file.

The visual mode permits the user to select the text as you would do with a mouse but using the keyboard instead of the mouse.

- v is used to select text, you can move using arrows.With V you can select entire lines at a time.
- y will “yank” the text into the buffer to be pasted later.
- With p or P you can past text
- d is used to cut and dd will cut an entire line

SSH Keys

What is SSH?

SSH is a software package and protocol for cryptographic, secure file transfer over a network that may or may not be secure. To get remote access to another computer, one can easily and securely set up a connection with the secure shell, or ssh, command. Mac and Linux can run ssh from their command lines, whereas for Windows PuTTY is a popular client running ssh protocols. This tutorial will only discuss operations working on Mac and Linux.

Types of public key algorithms for authentication keys:

- **rsa** - an old algorithm based on the difficulty of factoring large numbers. RSA is old and significant advances are being made in factoring, meaning the safety of using RSA may be questionable in the future.
- **dsa** - Digital Signature Algorithm. It is based on the difficulty of computing discrete logarithms; no longer recommended.
- **ecdsa** - a new Digital Signature Algorithm standarized by the US government, using elliptic curves, recommended for current applications.
- **ed25519** - this is a new algorithm added to OpenSSH. Support for it in clients is not yet universal, so it may not work across all platforms.

A basic example of the command is:

```
$ ssh remote_host
```

where *remote_host* here would be replaced by the IP address or domain name of the machine that you are trying to connect to.

If your username is required / different on the remote system, you can specify it by using this syntax:

```
$ ssh remote_username@remote_host
```

For example, to ssh into an Imperial computer you need your username and password and can use `ssh -XY username@login.hpc.imperial.ac.uk` or something similar, as detailed on the Imperial ssh gateway help site [https://www.imperial.ac.uk/admin-services/ict/self-service/research-support/rcs/support/getting-started/using-ssh/].

Install & Start

One way to install the open ssh server on the desktop machine is via sudo, a package for Unix-like computer operating systems giving the user permission to do certain things more easily, like installing a package:

```
$ sudo apt-get install openssh-server
```

You can check that the server is running with the command

```
$ sudo service ssh start
```

And output

```
start: Job is already running: ssh
```

You can check if the firewall is running with

```
$ sudo ufw status
```

and output:

```
Status: inactive
```

IP address

You can find your own IP address on Linux/Mac with the `ifconfig` command and look at the number after 'inet' that is not your localhost address (`127.0.0.1`). You can print the IP address with the following line:

```
ifconfig | grep "inet " | grep -Fv 127.0.0.1 | awk '{print $2}'
```

and you'll get something like:

```
193.52.232.22
```

When to use SSH keys

You don't need keys for ssh if your server is on the same wifi network as your computer, you can simply use your username, password, and the IP Address of the server to ssh into the machine. Keys come in unique pairs of a public key and a private key. These two keys have a special mathematical property that lets you prove you have the private key without showing what it is, a process handled by the protocol.

Public key authentication broadly works in these steps:

1. Generate your key pair.
2. Give a server the public key.

3. Later, whenever you want to authenticate, the server asks you to prove you have the private key that corresponds to the public key.
4. You prove you have the private key.

The SSH server and client programs take care of the authentication process and mathematical check for you.

How to setup SSH keys

1. *Generate SSH Keys*

To make an ssh key pair, all you have to do is invoke the generation command

```
$ ssh-keygen
```

this will give you an output like

```
Generating public/private rsa key pair.
```

The algorithm type is selected using the -t option and key size using the -b option. The following command illustrates choosing rsa for your key generation:

```
ssh-keygen -t rsa
```

You'll be prompted to choose the location to store the keys. The default location is good unless you already have a key. Press Enter to choose the default location.

```
Enter file in which to save the key (/Users/yourname/.ssh/id_rsa):
```

Next, you'll be asked to choose a password. Using a password means a password will be required to use the private key. It's a good idea to use a password on your private key.

```
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
```

After you choose a password, your public and private keys will be generated. There will be two different files. The one named id_rsa is your private key. The one named id_rsa.pub is your public key.

```
Your identification has been saved in /Users/yourname/.ssh/id_rsa.
Your public key has been saved in /Users/yourname/.ssh/id_rsa.pub.
```

You'll also be shown a fingerprint and "visual fingerprint" of your key. You do not need to save these.

```
The key fingerprint is:
d7:21:c7:d6:b8:3a:29:29:11:ae:6f:79:bc:67:63:53 yourname@laptop1
```

```
The key's randomart image is:
+--[ RSA 2048]----+
|                 |
|            . o  |
|       .   . * . |
|      . .   = o  |
|       o S . o   |
|      . . o oE   |
|     . .oo +.    |
|      .o.o.*.    |
|      ....= o    |
+-----------------+
```

2. *Move the public key to the remote machine*

To use public key authentication, the public key must be copied to a server and installed in an authorized_keys file. This can be conveniently done using the ssh-copy-id tool. Like this:

```
ssh-copy-id -i ~/.ssh/tatu-key-ecdsa user@host
```

Once the public key has been configured on the server, the server will allow any connecting user that has the private key to log in. During the login process, the client proves possession of the private key by digitally signing the key exchange.

Now you can test your connection by ssh into your remote server!

Links:

- Secure Shell Homepage: https://www.ssh.com/ssh/
- Imperial ssh details: https://www.imperial.ac.uk/admin-services/ict/self-service/research-support/rcs/support/getting-started/using-ssh/

File Management in Shell Scripting

· *Introduction to Shell Scripting*

“A shell in a Linux operating system takes input from you in the form of commands, processes it, and then gives an output. It is the interface through which a user works on the programs, commands, and scripts. A shell is accessed by a terminal which runs it. “ (https://www.guru99.com/introduction-to-shell-scripting.html). It is also important to keep in mind that shell scripts are interpreted but not compiled, i.e. there is no need to compile the shell script before executing it. Although there are many shell types that the operating system of a computer supports in this tutorial we are going to explore “bash scripting” due to its flexibility and ease in understanding.

· *Creating a Shell Script*

1. Create a file using an editor and name the script file with the extension .sh. This can be either done by directly creating a text document with the extension .sh or by opening the Linux command line navigating to the directory where your file to reside and typing “touch filename.sh”. This will create a blank document ready to be edited.

2. Start the script with #!/bin/bash. This will help understand the interpreter that your file is a bash shell script.

3. Write your commands.

· *Run the .sh file*

You can run the script from the bash/Linux terminal using several commands. The most common command is ‘bash’:

-- bash filename.sh

Commonly used is also the following command:

-- ./filename.sh At this point, insert the following command in the .sh file you have just created:

-- echo “My first shell scripting task!”

As we will see later, the command “echo” is simply used to print out a message to the standard output (i.e. command line). If you try to run your .sh file now containing both the “#!/bin/bash” header and the “echo” statement, then it is very likely that you will get a permission error that says (beware that it also possible that the following does not occur, if it does not then ignore the few following steps):

-- bash: ./filename.sh: Permission denied

In order to gain permission to edit the file and run it the “chmod” command is used as follows:

-- chmod -x filename.sh

Now if you retry running the .sh file it should now give the following output:

-- My first shell scripting task!

At this point we are able to edit the shell document at our needs. All the commands that you have learned in the previous section can be used in the shell script that you have just created. So, by now the shell script that you have written should look like something like this:

At this point, it would be good practice for you to use the commands that you have learned in the previous sections and write them in the shell script. This helps to get confident with this new way of writing to the command terminal and to reinforce what you already know. Note: the use of comments in bash shell scripting is done by just putting the character “#” in front of the text that you want to be the comment.

· *Redirecting output of a command to a file (cat, >, >>):*

While using the bash shell scripting is very important to handle the output to other files. Commands such as “ls”, in fact, by default redirect their outputs to the standard output which is the command line. We can also redirect the output of this commands to be something other than the standard output (e.g. a text file). This can be done either by the commands “>” and “>>”. The first one overwrites the contents of the “host” file where the output is redirected to, while the second one just appends the output to the current contents of the “host” file. It is also important to know that the “cat command” is used to print the content of a file to the standard output (i.e. command line). For instance:

-- ls > myfiles.txt -- echo "These are the contents of myfiles.txt:" -- cat myfiles.txt

This series of commands will then just redirect the output of “ls” to a file called “myfiles.txt”, prints a message regarding the contents of the host file to the command line and finally shows the contents of the same file. The output should look something like this:

-- My first shell scripting tutorial! -- These are the contents of myfiles.txt: -- myfiles.txt -- shell_1.sh

Note that the output may vary depending on whether the current working directory has other files other than “myfiles.txt” and “shell_1.sh”. Using the same commands, we can also redirect a simple line of text to another file:

-- echo "My Files" >> myfiles.txt -- cat myfiles.txt

Note that this time, using “>>” instead “>” the message “My Files" was just appended to the content of “myfiles.txt”. Ignoring the output from the previous example, the output should be as follows:

-- myfiles.txt -- shell_1.sh -- My Files

· *Redirecting standard input of a command to read from a file:*

The standard input of a command can be redirected such that it reads from a file instead of the standard input. This command is not widely used but is worth exploring it. Considering we want to count the number of lines, words and characters of a file. This is done through the command “wc” as follows:

First, we can use the existent file “myfiles.txt” and fill it with some text that can then be counted:

-- echo “Hi I am learning shell scripting” > myfiles.txt -- ls >> myfiles.txt

We can then try to use the command “wc” to the “myfiles.txt” as follows:

-- echo “Word count result:” -- wc < myfiles.txt

This should output something like the following:

-- Word count result: -- 4 9 69

As stated above the three numbers resulting from the operation are respectively the number of lines, words and characters of the content of the file o which “wc” is applied. It is worth noting that an almost identical output could have been reached if the command “wc myfiles.txt” was used with the exception that the output would also include the name of the file together with the values output of the command:

-- 4 9 69 myfiles.txt

· *Piping:*

Piping is a very powerful command in Linux shell scripting as it allows to carry out more than one task at the same time. This command also allows to connect the output of one program to the input of another This is allowed by the command “|” (called “pipe”) in the following way:

-- command1 | command 2

To better understand this idea one can introduce the commands “head” and “tail” which allow to print the first and the last ten lines of the content of a file respectively for example. For instance:

-- ls | head -- ls | tail

These two commands respectively print to the standard output the first and the last ten lines of the of the current working directory. In addition, when applying “-n” plus the number of lines, it will print the first n lines when applied for example to the command “head” e.g.:

-- ls | head -n 3

Same reasoning can be applied to the command “tail”. We can now pipe these two commands together and adding a wordcount command as follows:

-- ls | head -n 5 | tail – n 3 | wc

This piping will allow us to count the number of lines, words and characters of the last 3 lines taken from the first five lines of the output of ls (i.e. the current working directory).

· Grep:

The command “grep” is very useful as it allows the user to look for patterns inside a file. This command searches through the input file for an expression given and prints out (to the standard output by default) the lines that match the given expression. Let’s see how this works: Firstly, create a new text file and call in “grepfile.txt” for example. We can then write to it using the command echo as follows:

-- echo “This file contains many words” > grepfile.txt

-- echo “Words are fundamental to communicate” >> grepfile.txt

-- echo “Words are made of characters” >> grepfile.txt

-- echo “Words have to be used carefully” >> grepfile.txt

now we can use the grep command to print out all the lines that contain the word “words” in the file “grepfile.txt” as shown below:

-- echo “Grepped file: ”

-- grep “words” grepfile.txt

this should now output all the lines in the file since each line contain the given expression “words”. The “grep” command can also be used in combination with piping. It can in fact be used to search for all the bash files in our current working directory using the following: -- echo “All my .sh files: “

-- ls | grep “.sh”

At this point the output of this command could also include backup files (files with the character ‘ ~ ‘ at the end of its name. To not include these types of files, the following command has to be run: -- ls | grep “.sh$” This should now output exclusively the files with “.sh” extensions i.e. the bash files in the current working directory. Finally, the “grep” command can be used on multiple files at the same time; this is done by specifying the names of all the files you want to use the command on:

-- grep expression file1.txt file2.txt file3.txt …

· For loop: For loops are a useful tool used in every programming language. For bash scripting the syntax is as follows

-- for <expression>

do

<commands> …

done

there are many expressions that can be used, and the way they are written is similar to python. You can find various examples on this site

https://www.cyberciti.biz/faq/bash-for-loop/

· While loop: For loops are a useful tool used in every programming language. For bash scripting the syntax is as follows

-- while [<expression>]

do

<commands> … ù

Done

Any command between do and done is executed as long as the expression is true.

· If statement: If statements are probably used at least once in every code. For bash scripting the syntax is as follows

-- If [<expression>]

then

<commands> …

fi

Any command between then and fi will be executed only if the expression between the square brackets is true. An expression could be as simple as [x == 1] or very complicated, including -o (logical or) and -a (logical and) gates.

· Other loops and statements: To learn other kind of loops like switch… or more about the loops and statements above explained any online source. These are very basic operations therefore there are plenty of

sites online that are great. We suggest to look around online and have fun with your shell script, maybe try printing all the odd numbers between 1 and 100 or the names of the files in a directory containing the .sh .

· *Useful tips and commands:*

It could also be useful to rename a directory or a file using the shell scripting. This is simply done using the “mv” command, which has the following syntax/ expression type:

-- mv [filename] [newfilename]

In this case it could also be useful to rename all files in a directory as well. In order to carry this out, it is required to navigate to the desired directory that contains the files that we want to rename and using a for loop to rename the files you desire to rename. This works as follows:

-- cd folder

-- for f in *; do

-- mv “$f” “${f/filename/newfilename}”

-- done

Where the operator “*” means to take all the files in that directory. If instead we want to select all the files with a certain extension for example, we would type “*.extension”. This, applied to the above example would rename all the file with the specified extension. A similar approach can be used to delete files/directories using the “rm” command whose expression is works as follows:

-- rm filename

Therefore, similarly to what seen above, one could simply delete all the files in a directory with a specific extension. This would appear as follows:

-- cd folder

-- for f in *.extension; do

-- rm f

-- done

scp