Hillary 2020

Introduction to Linux Hilary 2020

Introduction to Linux

- All of the current ARC systems run an operating system called Linux. Whereas Microsoft Windows and Mac OS X place almost total emphasis on graphical interaction with the operating system, Linux encourages users to do lots of things from the "command line" or "prompt".
- Working from a command line has many benefits, especially when dealing with multiple files or performing complex operations on data, and once used to it, many users complain how slow and painful using graphical systems can be.
- it is different and many users find the learning curve off putting. Simple tutorials with Linux will help you.

Purpose of this tutorial

- To help users of the University of Oxford Advanced Research Computing facility to gain sufficient basic Linux knowledge and skills to be able to utilise our services.
- We also recommend that after this you next attend the HPC: Introduction to the Advanced Research Computing service course.

Command line

- ouit0578@login12(arcus-b)$ is an example of a linux prompt.
- commands can only be entered once we have the prompt.
- all commands give us an output, correct commands give us a response, wrong commands give us : command not found"

Most basic commands

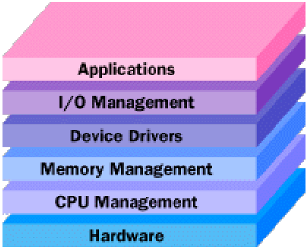
Command	Meaning
ls	content of current working directory
cd directoryname	Change directory
passwd	Change password for my user name
file and filename	Display file type ...
cat textfile	Shows content of text file
pwd	print [current] working directory
exit or logout	End this session
man and command Read	manual pages about command
info and command Rea	info pages on command
apropos string	the whatis database

Operating System

On a computer the operating system is the program that relays instructions to the various parts of the computer and makes sure that they are carried out.The operating system turns those instructions into something the computer can understand. Computer Instructions are sent

- from user typing commands at the command line
- from an application such as the program you run to perform your research
- from a graphical user interface

The Linux File System



The Linux operating system is built around the concept of a filesystem. This stores not only the operating system itself, it in fact stores everything for the whole system including your data and program files, files for commands, system and software configuration information, temporary workfiles, and various special files that are used to give controlled access to system hardware and operating system functions.

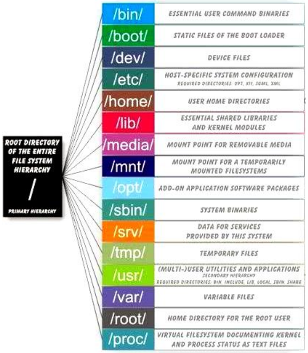
Linux characteristics

- Linux is case sensitive – In almost all circumstances a and A are considered to be different. **'ls' is a standard Linux command, while LS is not** and abc.dat is a different file from Abc.Dat.
- There are four types of files in a Linux filesystem: Files, Directories, Devices, Links.

Linux File Types

- Files: contain text, data, or program information. Filenames can contain any character except for '/' and be up to 256 characters long. However we strongly suggest you avoid characters such as **#,$,** as they have special meaning in Linux. Putting spaces in filenames also makes them difficult to manipulate, use the underscore instead.
- Directories containers that hold files, and other directories. These are an almost direct equivalent for Windows folders.
- Devices: To provide applications with easy access to hardware devices, Linux allows them to be used like ordinary files. There are two types of devices in Linux - block-oriented devices which transfer data in blocks (e.g. hard disks) and character-oriented devices that transfer data on a byte- by-byte basis
- Link: a pointer to another file. There are two types of links - a hard link to a file is indistinguishable from the file itself. soft link (or symbolic link) provides an indirect pointer to a file.

Linux Directory Structure

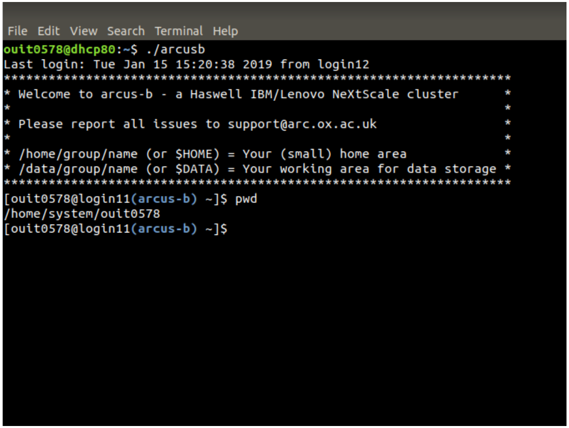


ROOT DIRECTORY OF THE ENTIRE FILE SYSTEM HIERARCHY
/
PRIMARY HIERARCHY
/bin/
ESSENTIAL USER COMMAND BINARIES
/boot/
STATIC FILES OF THE BOOT LOADER
/dev/
DEVICE FILES
/etc/
HOST-SPECIFIC SYSTEM CONFIGURATION
REQUIRED DIRECTORIES: OPT, X11, SGML, XML
/home/
USER HOME DIRECTORIES
/lib/
ESSENTIAL SHARED LIBRARIES AND KERNEL MODULES
/media/
MOUNT POINT FOR REMOVABLE MEDIA
/mnt/
MOUNT POINT FOR A TEMPORARILY MOUNTED FILESYSTEMS
/opt/
ADD-ON APPLICATION SOFTWARE PACKAGES
/sbin/
SYSTEM BINARIES
/srv/
DATA FOR SERVICES PROVIDED BY THIS SYSTEM
/tmp/
TEMPORARY FILES
/usr/
(MULTI-)USER UTILITIES AND APPLICATIONS
SECONDARY HIERARCHY
REQUIRED DIRECTORIES: BIN, INCLUDE, LIB, LOCAL, SBIN, SHARE
/var/
VARIABLE FILES
/root/
HOME DIRECTORY FOR THE ROOT USER
/proc/
VIRTUAL FILESYSTEM DOCUMENTING KERNEL AND PROCESS STATUS AS TEXT FILES

The directory structure in Linux is a tree like structure with the so called **"root"** directory at the bottom, with other directories branching off it.

Basic Directory and File handling commands

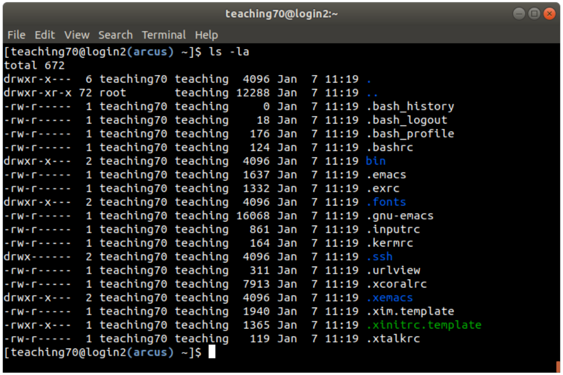
To use the system we open up a terminal or log into an ARC account. At the prompt if we type pwd (print working directory) we see the full absolute path to your current location in the filesystem. for example **/home/ouit0578** or /home/teaching01



```
ouit0578@dhcp80:~$ ./arcusb
Last login: Tue Jan 15 15:20:38 2019 from login12
*******************************************************************
* Welcome to arcus-b - a Haswell IBM/Lenovo NeXtScale cluster     *
*                                                                 *
* Please report all issues to support@arc.ox.ac.uk                *
*                                                                 *
* /home/group/name (or $HOME) = Your (small) home area            *
* /data/group/name (or $DATA) = Your working area for data storage *
*******************************************************************
[ouit0578@login11(arcus-b) ~]$ pwd
/home/system/ouit0578
[ouit0578@login11(arcus-b) ~]$
```

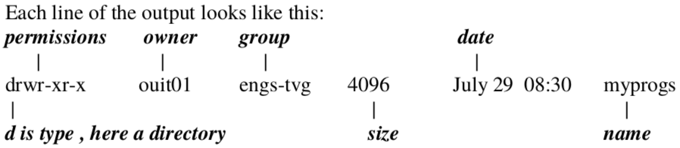
Basic Directory and File handling commands

- list directory **ls** lists the contents of a directory. If we don't specify a directory, then the contents of the current working directory are displayed.
- list all files including files and directories that begin with a dot and are hidden **ls -la** : ls doesn't show all the entries in a directory, files beginning with a dot usually contain configuration information and should not be changed under normal circumstances. If we want to see all files, ls supports the –a (for all) option or "flag": ls -la



```
[teaching70@login2(arcus) ~]$ ls -la
total 672
drwxr-x---  6 teaching70 teaching  4096 Jan  7 11:19 .
drwxr-xr-x 72 root       teaching 12288 Jan  7 11:19 ..
-rw-r-----  1 teaching70 teaching     0 Jan  7 11:19 .bash_history
-rw-r-----  1 teaching70 teaching    18 Jan  7 11:19 .bash_logout
-rw-r-----  1 teaching70 teaching   176 Jan  7 11:19 .bash_profile
-rw-r-----  1 teaching70 teaching   124 Jan  7 11:19 .bashrc
drwxr-x---  2 teaching70 teaching  4096 Jan  7 11:19 bin
-rw-r-----  1 teaching70 teaching  1637 Jan  7 11:19 .emacs
-rw-r-----  1 teaching70 teaching  1332 Jan  7 11:19 .exrc
drwxr-x---  2 teaching70 teaching  4096 Jan  7 11:19 .fonts
-rw-r-----  1 teaching70 teaching 16068 Jan  7 11:19 .gnu-emacs
-rw-r-----  1 teaching70 teaching   861 Jan  7 11:19 .inputrc
-rw-r-----  1 teaching70 teaching   164 Jan  7 11:19 .kermrc
drwx------  2 teaching70 teaching  4096 Jan  7 11:19 .ssh
-rw-r-----  1 teaching70 teaching   311 Jan  7 11:19 .urlview
-rw-r-----  1 teaching70 teaching  7913 Jan  7 11:19 .xcoralrc
drwxr-x---  2 teaching70 teaching  4096 Jan  7 11:19 .xemacs
-rw-r-----  1 teaching70 teaching  1940 Jan  7 11:19 .xim.template
-rwxr-x---  1 teaching70 teaching  1365 Jan  7 11:19 .xinitrc.template
-rw-r-----  1 teaching70 teaching   119 Jan  7 11:19 .xtalkrc
[teaching70@login2(arcus) ~]$ 
```

Basic Directory and File handling commands

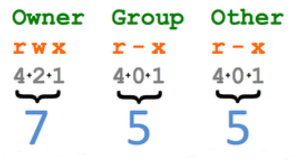


Each line of the output looks like this:

```
permissions   owner    group                  date
    |           |        |                      |
drwr-xr-x     ouit01   engs-tvg   4096    July 29  08:30    myprogs
|                                   |                          |
d is type , here a directory       size                       name
```

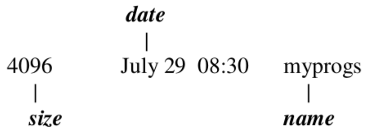
A lot of information here! At this stage the most important are the name of the file, the date it was last modified, and its size in bytes. For completeness the rest are type : a single character 'd' (directory), '-' (ordinary file), 'l' (symbolic link), 'b' (block device) or 'c' (character device) and then we have Permissions.

File Permissions



- There are 9 permission characters, describing 3 access types given to 3 user categories.
- The 3 users categories are the user who owns the file, users in the group that the file belongs to and other users (the general public).
- The three access types are **read ('r'), write ('w') and execute ('x'). An 'r', 'w' or 'x'** character means the corresponding permission is present; a '-' means it is absent. Links refers to the number of filesystem links pointing to the file/directory
- Owner is usually the user who created the file or directory.
- Group: a collection of users allowed to access the file according to the group access rights specified in the permissions field. On ARC machines all members of a project are in one group.
- Other means all user who can login to ARC systems

File Permissions continued



- size is the length of a file, or the number of bytes used by the operating system to store the list of files in a directory
- date is the date when the file or directory was last modified. The -u option display the time when the file was last accessed (read). name is the name of the file or directory.

We should also mention 'sticky bit' a permission bit that is set on a file or a directory that lets only the owner of the file/directory or the root user to delete or rename the file, it appears as s or t.

More on ls

- ls (like many Linux commands) has many more flags, type ‘man ls’ or ‘info ls’ to see the options – this also tells use the standard Linux help commands. When using info pressing space means display the next page, and q means quit to the command prompt.
- when you first login to your user id you will not have many files in your directory! A simple way to create a file is the touch command, which will create a new file if one with the given name does not exist.
- touch my.txt. We can use ls -l to find out how big the file is.
- touch change the time stamp of a file. If we touch the file for a second (or third, fourth...) we will see with ls -l the time changing.
- touch is not actually that useful for creating files, we’ll see better ways soon!

Create and change directories

We create a new directory within mydata called results and change into it, and use pwd again. Commands relating to directories

- mkdir mydata and we use ls to find out what has happened.
- cd is the change directory command
- pwd displays current directory

Copy and move

- Copy a file **cp** and **mv** and move, (i.e. rename, a file). First we make sure we are in the right directory and we have the correct permissions using ls -l or ls -la then use the following command :
- To copy : **cp my.txt myold.txt** Now try the command **mv myold.txt myveryold.txt**. Using ls can we can show the difference between what cp and mv do. With mv the original file is no longer there.
- To delete a file use **rm**. BEWARE! By default Linux won't ask if we want to do this, it will just do it. Thus to get rid of the file **rm myveryold.txt** we always use ls to see what has happened. Experienced command line users prefer not being asked, however if we want to be careful we can use the –i flag to rm which will ask if the user wants to remove the file: **rm -i myfile** remove 'myfile'.
- By default rm won't remove a directory, use rmdir command which will, but only as long as the directory is empty.
- To delete a directory and its contents use **rm -r directoryname**

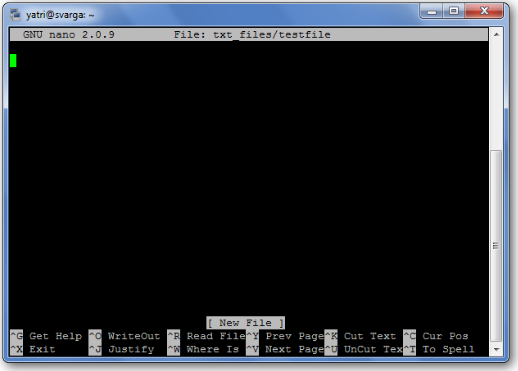
Summary so far and Editors

Now you should understand how

- how the filesystem is structured
- how to move about in it
- some of the most common file manipulation commands. We don't yet know how to give the files any content.
- The most common way to give a file content is to use an editor – a special program that allows us to type into a file, and change a file's contents.
- There are many linux editors (e.g. emacs, gedit, nano). Here we will illustrate use of a fairly simple one, nano. Others are broadly similar but the details will vary how the filesystem is structured how to move about in it.

Editor Nano

We would start editing by typing : **nano my.txt**. If the file does not exist it will be created. You would then see a screen like the image below which displays the file's contents (currently nothing).

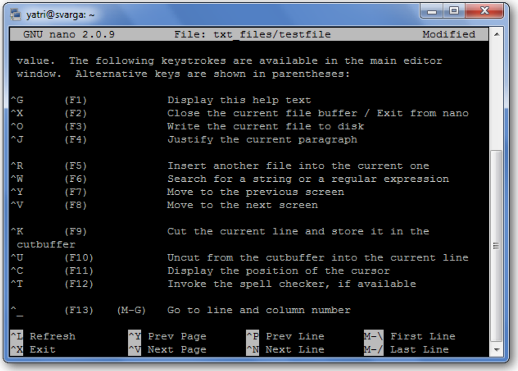


```
  GNU nano 2.0.9             File: txt_files/testfile

                                [ New File ]
^G Get Help  ^O WriteOut  ^R Read File^Y Prev Page^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is ^V Next Page^U UnCut Tex^T To Spell
```

More on editor nano

You can then hit Ctrl+G to bring up the Help documentation and scroll down to see a list of valid shortcuts.



```
 GNU nano 2.0.9          File: txt_files/testfile                 Modified

 value.  The following keystrokes are available in the main editor
 window.  Alternative keys are shown in parentheses:

^G      (F1)            Display this help text
^X      (F2)            Close the current file buffer / Exit from nano
^O      (F3)            Write the current file to disk
^J      (F4)            Justify the current paragraph

^R      (F5)            Insert another file into the current one
^W      (F6)            Search for a string or a regular expression
^Y      (F7)            Move to the previous screen
^V      (F8)            Move to the next screen

^K      (F9)            Cut the current line and store it in the
 cutbuffer
^U      (F10)           Uncut from the cutbuffer into the current line
^C      (F11)           Display the position of the cursor
^T      (F12)           Invoke the spell checker, if available

^_      (F13)   (M-G)   Go to line and column number

^L Refresh          ^Y Prev Page        ^P Prev Line        M-\ First Line
^X Exit             ^V Next Page        ^N Next Line        M-/ Last Line
```

We use nano to provide some contents for the file my.txt in my home directory and save the changes. If we want to see how big is this file we use $ls –l to find out.

More Commands

Now we can make files with contents it would be nice to display them at the command line without having to open nano every time. Two useful commands to do this

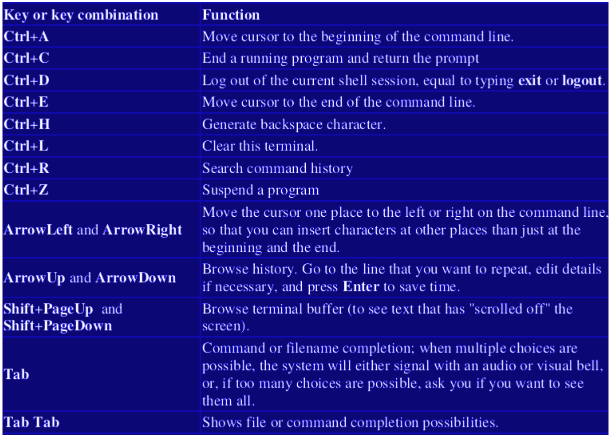
- To just display the file “as is” we use **cat**. Later we can create a file and try cat my.txt. cat is good for short files
- If we use cat for files that take more than one screen to display, the top will disappear probably before we can read it! Command **less** will let us see content one page at a time. See how less my.txt works. To quit any of these commands use q, it will take me to the command prompt, just as it did for info.
- To show the top and the bottom of the given file respectively we use **head and tail**.

Command Line Short Cuts

You will have noticed now that often we repeat very similar commands and it would be useful to be able to repeat, or slightly change, previous commands without having to retype them.

- Hitting **the up arrow** will display the command we just used.
- Using **the left and right keys**, we can move around in the command change it as required.
- Hitting **the up arrow** again will go back one further command. You have a complete history of your commands which is searchable an editable.

list of useful short cuts at the command line



Key or key combination	Function
Ctrl+A	Move cursor to the beginning of the command line.
Ctrl+C	End a running program and return the prompt
Ctrl+D	Log out of the current shell session, equal to typing **exit** or **logout**.
Ctrl+E	Move cursor to the end of the command line.
Ctrl+H	Generate backspace character.
Ctrl+L	Clear this terminal.
Ctrl+R	Search command history
Ctrl+Z	Suspend a program
ArrowLeft and **ArrowRight**	Move the cursor one place to the left or right on the command line, so that you can insert characters at other places than just at the beginning and the end.
ArrowUp and **ArrowDown**	Browse history. Go to the line that you want to repeat, edit details if necessary, and press **Enter** to save time.
Shift+PageUp and **Shift+PageDown**	Browse terminal buffer (to see text that has "scrolled off" the screen).
Tab	Command or filename completion; when multiple choices are possible, the system will either signal with an audio or visual bell, or, if too many choices are possible, ask you if you want to see them all.
Tab Tab	Shows file or command completion possibilities.

More shortcuts at the Command line

- If we want to change into a directory with a very long name, we don't need to type that long name, on the command line, we use **Tab**: cd dir (first 3 letters of your long name), then we press Tab and the shell completes the name provided no other files starts with the same three characters.
- If there are no other items starting with "d", then we can just type cd d and then Tab. If more than one file starts with the same characters, the shell will signal this, upon which we can hit **Tab** choice. This refers to hitting Tab twice with short interval, and the shell presents the choices we have.
- we have 3 directories starting with the same letters: gnome-run1 gnomics gno-test if we use 'cd gno' after the first three characters and hit **Tab**, the shell completes the directory name, showing all 3 options. If we add an m and hit tab again we will get gnome-run1 gnomics as these are the only 2 of the 3 that now match what we have at the command line.

Using ARC system

At this point we have reached a stage whereyou should have enough knowledge to do most of your work on ARC systems! However there are a few other topics which, while not absolutely necessary, can be useful to aware of, especially redirection which we consider below. These begin to illustrate the power of the command line for complex operations on multiple files.

Specifying Multiple Files and Wildcards

So far we have worked with one file at a time. However Linux allows us to work with many files at once, and many commands can be applied to multiple files at once.

- character replacement: it is boring to have to type out all these file names. Instead we can look for multiple filenames using special pattern-matching characters, often called wildcards.
- use simple rules '?' matches any single character in that position in the filename; '*' matches zero or more characters in the filename; a '*' on its own will match all files; '*.*'matches all files containing a '.'; characters enclosed in square brackets ('[' and ']') will match any filename that has one of those characters in that position.

Specifying Multiple Files and Wildcards- Examples

- ls ??? shows all the files in the current directory that have exactly 3 characters in their name .
- ls ?ell? matches any five-character filenames with 'ell' in the middle. or he* matches any filename beginning with 'he', e.g. hello, help, heterogeneous but NOT Henrietta - Remember Linux is case sensitive!
- ls [m-z]*[a-l] matches any filename that begins with a letter from 'm' to 'z' and ends in a letter from 'a' to 'l'. Thus [Hh]e* matches all of hello, help, heterogeneous and Henrietta. Note that the Linux shell performs these expansions (including any filename matching) on a command's arguments before the command is executed

Quoting

A number of special characters (e.g. *„$ ..) are interpreted in a special way by the shell. In order to pass arguments that use these characters to commands directly (i.e. without the filename expansion we have just seen etc.), Linux uses special quoting characters which forces them to be interpreted as-is rather than in any special way. There are three forms of quoting:

- inserting a \in front of the special character.
- using double quotes ” around arguments to prevent most expansions.
- using single forward quotes ’ around arguments to prevent all expansions. In the directory with 1.txt we try the commands: ls *.txt and ls "*.txt”
- There is a fourth type of quoting in Linux. Single backward quotes ‘ are used to pass the output of a command as an input argument to another.

Finding Files

If I don't know the exact location of a file there are ways of finding it using the find command. **find directory –name filename -print .** find will look for a file called filename any part of the directory tree rooted at directory.

- find . -name "*.txt" -print . This will search all user directories for any file ending in ".txt" and output any matching files (with a full absolute or relative path). Here the quotes (") are necessary to avoid filename expansion
- find can in fact do a lot more than just find files by name, as usual **man find** will provide information. A particularly useful form of this is find . –name "*.txt" –print. Note the period (.) after the find command – this is Linux shorthand for the current directory. Thus this will find all files ending in .txt in the current directory and all directories underneath it.

Finding text in Files

The command grep (general regular expression print) searches files for lines that match a given pattern (the technical term for here is “regular expression”). For example

- We create a file start.txt and enter the words Twinkle, Twinkle star and twinkle
- The command **grep Twinkle star.txt** will print out all lines that contain the text Twinkle (but not twinkle - remember Linux is case sensitive) in the file star.txt.
- We have created a text file using nano and want to find a way so that grep does find all lines with both Twinkle and twinkle? .
- We use **man grep** and find that grep -i twinkle will ignore case distinctions
- Let’s say we want to find every shell script (for us a file whose name ends in .sh) below the current directory that contains the text SBATCH. We use **grep SBATCH ‘find . -name ”*.sh” -print‘.**This searches all shell script files in the directory tree below the current directory for lines containing ”SBATCH”.

Sorting , redirecting Files

- The command **sort** sorts lines contained in a group of files alphabetically (or numerically if the -n flag is specified) numerically. To outputs the sorted concatenation of files input1.txt and input2.txt use sort file1.txt file2.txt.
- redirect - Our commands are getting increasingly complex. Sometimes it would be nice if we could save the output to a file. Technically every command writes its output to something called Standard Output. By default standard output in our terminal is going to the terminal. It is possible to change this so that standard output goes to a file instead of the display by use of the >character.

Redirecting Input and Output

- We use **echo** to display line of text/string that are passed as an argument will write Hello to the screen.
- To write the word Hello to a file called hello.txt we use **echo "Hello" >hello.txt** . Be a little careful as if hello.txt already existed it would overwrite it, and as we are using Linux we will get no warning.
- We can overwrite or add text **echo "Bye" >hello.txt**. What are the contents of the file now? If we want to avoid this use >>to append to a file.
- To take the input from Standard In, which is the keyboard **echo "Hello again" hello.txt** .

More on Redirecting

To redirect this as an input use >. This will cause a command to read its input from a given file. Redirection is especially important when we run our jobs in Batch mode, which is the main way of using the ARC systems, and the main topic you will learn about in the Introduction to ARC Service course.In such systems we will find programs are running in a way that does NOT allow us to type commands into it, and we are not always connected to a screen. In this case I/O redirection may be the only way to allow your program to read its input!

Pipes

We saw how to redirect the output of one command into a file. It is also often very useful to redirect the output so that it acts as the input

- To see example of a pipeline we will try: ”cat listfile |sort |uniq”
- We create this list file as a list of 10 names, some duplicated.
- We can use cat and sort to generate a sorted list of unique names in the file
- output of the command goes to the standard output.
- We can save the output of the sort command. Note the difference between >and <, the former redirects standard output of a command to a file, the pipe redirects the standard output of one command to the standard input of another command.
- Sort: we know sorts its standard input into alphabetic order and writes that to standard output. This output, because of the second pipe, acts as the input for the uniq command.
- uniq This is a new one – it removes all neighbouring duplicate lines in a file, and thus gives us our desired result.

Copying files - creating tar files

- tar backs up entire directory structures and files into a single archive file. **tar -cvf archivename listoffiles** to be archived where archive name will usually have a .tar extension.
- example tar –cvf myfiles.tar will create a file called myfiles.tar that contains all files in the current directory and all directories below it (note the . at the end, do you remember what this means in Linux?) In the above c=create,v=verbose (output filenames as they are archived), and f=file.
- To list the contents of a tar archive, we can use the tabulate option: tar -tvf archivename
- To extract files from a tar archive, use: **tar -xvf archivename** This can be very useful for transferring many files between machine – use tar to create a single file containing everything we want, transfer that single file, and then extract it on the remote machine. Note we will want to do this on ARC as any files that are not used for 6 months are under threat of deletion!

File compression

Tar files can get very big and so can take a long time to transfer, in which case a compression utility like gzip can be very useful.

- To compress files, use:**gzip filename**
- To reverse the compression process, i.e. once on the home machine, we use:**gunzip -d filename**

Shells and Shell scripts

So far everything would have been typed everything into a terminal. However we can store commands in a file and then run that file, effectively generating a new Linux command from the operations given in the file. This is called a **shell script**, a shell being the base interpreter that is sorting out what all the commands we use actually mean. Technically we have been using the bash shell, but there others. However at this level it makes little difference which one we use. Note the following when storing commands in a file

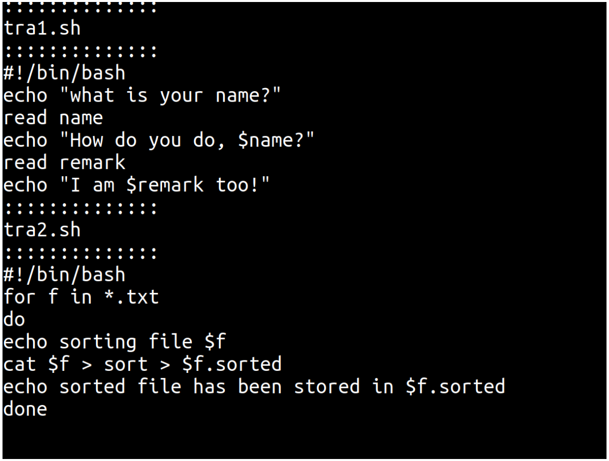
- 1. How to give execute permission to the file
- 2. How to run the file
- 3. How to make sure the commands in the file are interpreted as we want them to be.

More on Shells and Shell scripts

- 1. For executable files we need **Execute permission** . We revise the output generated by ls –l above each file has a set of permissions. One of these permissions tell us if the file can be executed or run. By default files cannot be run under Linux, this protects us against accidentally running something which shouldn't be, or worse running something maliciously installed on your system. However as we want to run our file we need to add the execute permission. You do this via 'chmod +x filename' which modifies a file called ¡filename¿ to have execute permission.
- 2.To run a file: the simplest way is to be in the same directory and then type **./filename**. This means try to run a file called filename in the current directory – remember what . in Linux means. To know more about this we have to look at the PATH environment variable which is touched on in one of the questions below, but to get started this simple recipe will do.
- 3.To make sure that the script is interpreted by the bash shell we should put #!/bin/bash as the very first line in the shell script. This sets the so called magic number for the script, and ensures that bash is used to interpret it, see https://stackoverflow.com/questions/8967902/why-do-you-need-to-put-bin-bash-at-the-beginning-of-a-script-file for more details.

Example of Shells and Shell scripts

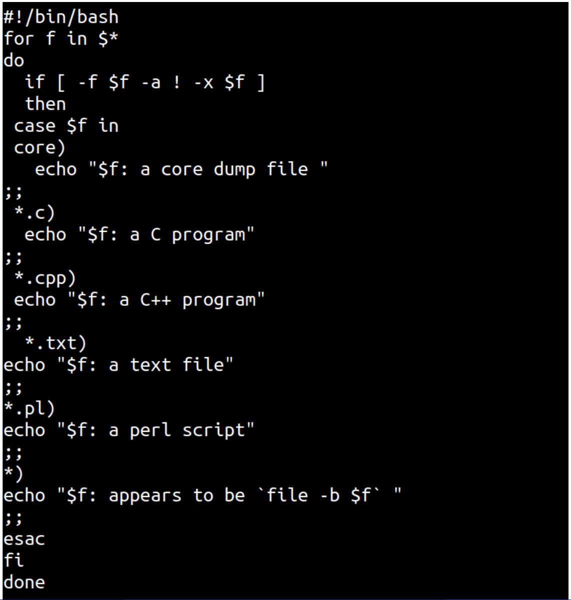
Simple examples of a shell script



```
::::::::::::::
tra1.sh
::::::::::::::
#!/bin/bash
echo "what is your name?"
read name
echo "How do you do, $name?"
read remark
echo "I am $remark too!"
::::::::::::::
tra2.sh
::::::::::::::
#!/bin/bash
for f in *.txt
do
echo sorting file $f
cat $f > sort > $f.sorted
echo sorted file has been stored in $f.sorted
done
```

Example of Shell Programming

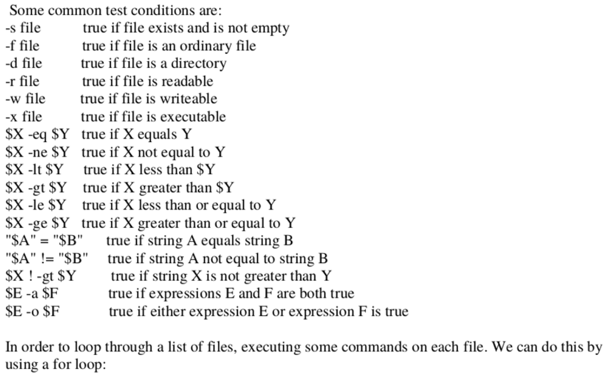
Example of a slightly more complicated Example.



```
#!/bin/bash
for f in $*
do
  if [ -f $f -a ! -x $f ]
  then
 case $f in
 core)
   echo "$f: a core dump file "
;;
 *.c)
  echo "$f: a C program"
;;
 *.cpp)
 echo "$f: a C++ program"
;;
  *.txt)
echo "$f: a text file"
;;
*.pl)
echo "$f: a perl script"
;;
*)
echo "$f: appears to be `file -b $f` "
;;
esac
fi
done
```

More Advanced Shell Programming

Here we very briefly mention a few more of the facilities provided by the Linux shell – don't worry if you don't understand them, none of the below is required to be a competent user of the ARC service, though some knowledge, especially of variables, may well be useful.



Some common test conditions are:

-s file	true if file exists and is not empty
-f file	true if file is an ordinary file
-d file	true if file is a directory
-r file	true if file is readable
-w file	true if file is writeable
-x file	true if file is executable
$X -eq $Y	true if X equals Y
$X -ne $Y	true if X not equal to Y
$X -lt $Y	true if X less than $Y
$X -gt $Y	true if X greater than $Y
$X -le $Y	true if X less than or equal to Y
$X -ge $Y	true if X greater than or equal to Y
"$A" = "$B"	true if string A equals string B
"$A" != "$B"	true if string A not equal to string B
$X ! -gt $Y	true if string X is not greater than Y
$E -a $F	true if expressions E and F are both true
$E -o $F	true if either expression E or expression F is true

In order to loop through a list of files, executing some commands on each file. We can do this by using a for loop: