Lab Week02 - Part I

PTA Linux Series

Shell Commands

Professional Training Academy Linux Series

Commands: Manipulation

- **cp** : copy a file
  - To copy a file you need to give a source and then a destination
  - e.g. to copy the file `poetry.txt` to `poetry_backup.txt`, type:

```
cp poetry.txt poetry_backup.txt
```

- **cp –r** : copy a *directory*
  - The **`-r`** stands for recursive which copies every file within a directory
  - e.g. to copy the directory `course_work` to `course_work_backup`, type:

```
cp –r course_work course_work_backup
```

PTA Linux Series

Commands: Manipulation

- **mv**: move a file or directory
  - As with copy, the move command also needs a source and a destination
  - e.g. to move the `course_work` directory to `new_course_work`, type:

```
mv course_work new_course_work
```

  - **Note:** To **rename** a file or directory in Linux, use the mv command

PTA Linux Series

Commands: Removal

- **Warning**: Removing files and directories is very easy and you may not be asked to confirm. Be very careful!

- **rm**: remove a file
  - e.g. to remove a file called `poety.txt`, type:

```
rm poetry.txt
```

- **rm –r**: remove a directory
  - e.g. to remove a directory named `course_work_backup`, type:

```
rm –r course_work_backup
```

PTA Linux Series

Commands: Removal

- **rmdir**: remove an empty directory
  - **rmdir** can be used as a check. If you want to be sure that the directory you are deleting is empty.

```
rmdir my_empty_directory
```

You can also remove a directory (empty, or not) with the command

rm -r

Which tells **rm** to remove **recursively** the file and any of its contents.

In UNIX a directory is just a special type of file. Just as a text file may contain words, a directory (file) contains other files.

Commands: Searching

- **find**: search for a file or directory
  - e.g. to search for a file named `project_results` within the current directory, type:

```
find ./ -name project_results –print
```

- to find all of the .cpp files in my entire directory modified in (or within) the last 2 days

```
find ./ -name '*.cpp' -mtime -2 –print
```

- **locate**: search for a program
  - e.g. to search for the location of the `gedit` text editor, type:

```
locate gedit
```

PTA Linux Series

Commands: Searching

- **which**: print the location of a program
  - e.g. to find out where the `ls` command is stored in the filesystem, type:

```
which ls
```

PTA Linux Series

Commands: Searching

- **grep** – search for regular expressions in files
  - e.g. to search for the word "and" in the `poetry.txt` file, type:

```
grep and poetry.txt
```

  - **Note**: If the word is not found within the poetry.txt file, grep will not print anything. However, if the word is found, grep will print the line where the word was found

- To search for the word "and" in all files in **all** directories (recursively), type:

```
grep –r and *
```

PTA Linux Series

Commands: Searching

- To make the search **case-insentive**, type:

```
grep –i and poetry.txt
```

- To search for the word “and” **inside** words, type:

```
grep *and* poetry.txt
```

- You can also search for **strings**, e.g.

```
grep “to be or not to be” poetry.txt
```

```
- search all sub-dirs for C files with 'open'
   find . -name '*.c' -exec grep open {}\;-print
```

PTA Linux Series

Commands: Help

- **man**: lookup a manual page for a command
  - **Note**: All commands in Linux have separate manual pages. You can use these to figure how to use a command or to find extra options. These options are called **arguments**. For example, when we used ls –l, the **–l** was the argument to the ls command
  - e.g. to look up the manual page for `ls`, type:

```
man ls
```

- Use the **space bar** to browse page by page
- To quit type **q**

PTA Linux Series

Commands: Help

- To **search** for a word within the manual pages, type forward slash **/**
  - The bottom left of the man page will then allow to you type a search word
  - All occurrences of the word will be highlighted
  - You can type **n** (for next) or **p** (for previous) to move to the next or previous occurrence

Exercises

1) Change into your projects directory and copy the **poetry.txt** to **poetry_backup.txt**
2) Copy your projects directory to a directory called **projects_backup**
3) Rename the **projects_backup** directory as **projects_completed**
4) Change into the **projects_completed** directory and delete the **poetry_backup.txt** file
5) Search the **poetry.txt** file for the word **“tHe”** or any other mixed case word in your file
6) Lookup the manual page for the copy command and search for the word recursive
7) Use the 'find' command to find all your .cpp files that were modified between 2 and 5 days ago.

Lab02 – Part II

PTA Linux Series

The Shell Environment

Professional Training Academy Linux Series

Shell Types

- There are many types of shells available on Linux. Most of which have a history from UNIX
- To check which types of shells are available on your system, type:

```
cat /etc/shells
```

  - Examples include: `bash`, `sh`, `ash`, `bsh`, `tcsh`, `csh`, `ksh`, and `zsh`
  - You can switch shell by typing it's name. Although there would be very few reasons for you to need to do this

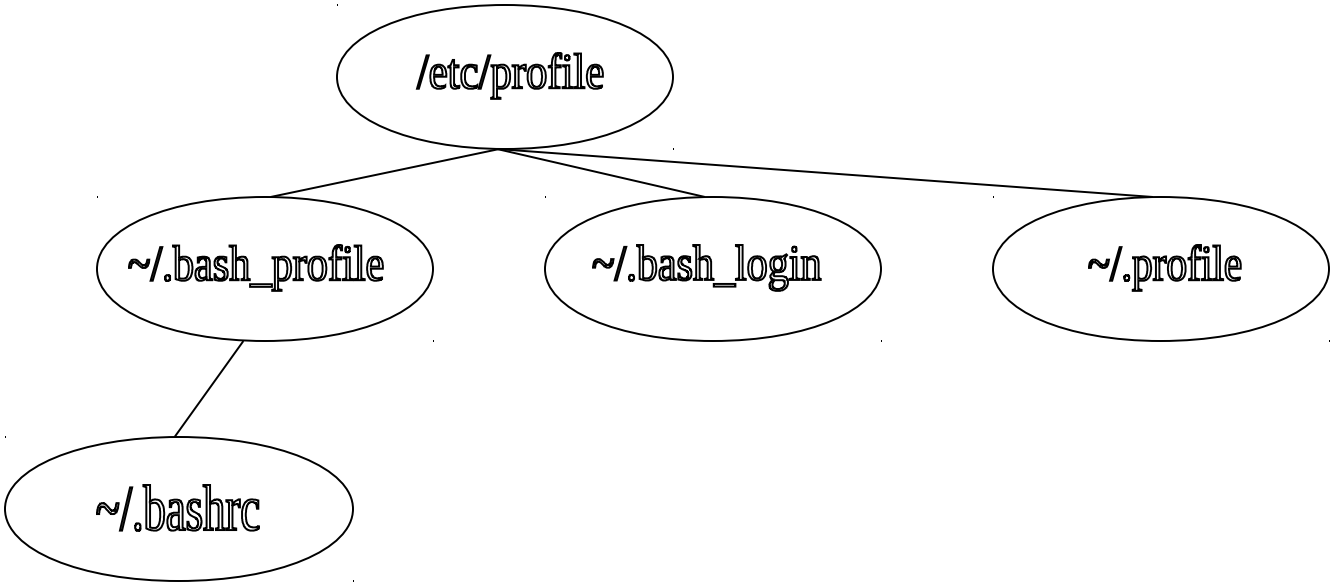
Shell Types

- The most popular shells are:
  - **`bash`**: Bourne Again Shell (we are using this shell)
  - **`tcsh`**: TC Shell
  - **`ksh`**: Korn Shell

- The shell is responsible for:
  - Parsing the command line and input
  - Executing programs
  - Evaluating special characters, e.g. *
  - Managing pipes, redirection, signals, and background processes

Shell Configuration Files

- The bash shell looks for a series of configuration files when starting up



PTA Linux Series

Shell Configuration Files

- The system wide configuration file is in `/etc/profile`
- As a user, you can customise your own shell using one of the hidden configuration files in your home directory, for example, `~/.profile`
- **Note**: Hidden files are any files beginning with a full stop `.*`
- These files are the place to store your environmental variables (explained later) or any other shell customisations you wish to make

PTA Linux Series

Command Line Input

- If you have typed in a long command and notice a typo at the start of the line, you may use **`ctrl-a`** to return to the start of the line
- Likewise, you may use **`ctrl-e`** to return to the end of the line
- The shell also supports auto-completion of command and file names
  - Type in the first few letters of a command and use **`tab`** to auto-complete the name
  - If there are multiple commands beginning with these letters, the shell will print back a list which you can choose from

Command Line Input

- The shell also records your command history
- You can access your previous commands using the **`up arrow`** and the **`down arrow`**
- A useful shortcut is to use the exclamation mark which looks up matches of your previous commands
- For example, if you had previously edited a file as follows: `emacs reallylongfilename`, you could use the following shortcut:

```
!emacs r
```

PTA Linux Series

Executing Programs

- If you execute a program, such as emacs from your shell, you will notice that you cannot type anymore commands until emacs is finished

- To avoid this, you can tell the shell to run any program such as emacs as a background process using the ampersand sign ***&***

```
emacs&
```


Executing Programs in your PATH

- The bash shell uses the **$PATH** environment variable to locate commands

- If you cannot run a program, you can echo the PATH variable as a check, for example:

```
echo $PATH
```

- To add a directory to your path, separate it by a colon, for example:

```
export PATH=$PATH:/my/cpp/programs:
```

PTA Linux Series

Creating an Alias for your Programs

- The bash shell allows you to create an **`alias`** or a shortcut name for programs
- The command **`alias`** will return a list of defined aliases in your shell
- Aliases can be useful for shortcuts, typos, and adding extra options, for example:

```
alias e=emacs
```

```
alias emcas=emacs
```

```
alias ls='ls -lh'
```

PTA Linux Series

Executing Programs with Environment Variables

- Environment variables can be set from the terminal or the shell configuration file
- For example, to set-up your proxy in the bash shell, type:

```
export http_proxy=http://student-proxy.ul.ie
```

- The **`export`** command sets the value of environment variables
- You will need to put this line into your shell configuration file to make it more permanent, e.g. **`~/.profile`**

Executing Programs with Environment Variables

- You can check the value of a variable using the **`echo`** command and placing a dollar sign before the variable, for example:

```
echo $http_proxy
```

- You can list the complete set of environment variables using the **`printenv`** or the **`env`** command

Special Characters

- Special characters will be evaluated by the shell
- For example, if you want to search all files in your current directory for the word "`password`", type:

```
grep password *
```

- The wildcard character (*) is evaluated by the shell and expanded to mean all file occurrences found

Special Characters

- Other special characters can be combined to form *regular expressions.* Examples of frequently used special characters include:

- **.** : Match a single character

```
grep ‘c.e’ filename
```

- ^ : Beginning of line

```
grep ‘^coffee’ filename
```

- **$** : End of line

```
grep ‘coffee$’ filename
```

- **\w** : Alphanumeric character [a-zA-Z0-9]

```
grep ‘c\w*e’ filename
```

`\w*` matches 0 or more

Redirecting Output

- The shell allows us to redirect the **output** of one command to the **input** of another command
- For example, if we do `ls` in a directory with hundreds of files it is difficult to keep track of all the files. However, the shell will let us take the output of the `ls` command and send it to a more useful viewing command such as `less`
- We can link the two commands using the **pipe** operator as follows:

```
ls | more
```

- The output of the command on the left becomes the input for the command on the right hand side

Redirecting Output

- The shell will also allow us to redirect the output to a file, which can be useful for creating log files
  - e.g. to save all the listed files to a logfile, type:

```
ls > logfile
```

- A double arrow operator can also be used for appending, e.g.

```
ls >> logfile
```

- Input can be taken from sources using the opposite arrow, e.g.

```
mail me@ul.ie < test_results
```


Shell Customisation – Prompt Messages

- In order to customise your shell, you need to alter the *Prompt String 1* environment variable, which is called **PS1**
- For example, if you wish to only have a right arrow for your prompt, type:

```
export PS1=“>”
```

- Or any other message:

```
export PS1=“beware of my shell >”
```

- Special characters can also be used, for example:
  - **\t** : current time in HH:MM:SS
  - **\w** : current working directory
  - **\u** : the current user
  - **\h** : the hostname
  - **\a** : the ASCII bell character

Shell Customisation – Colouring Prompts

- To add colour to your prompt, we add a numeric colour value between **`\e[`** and **`m`**. Numeric codes are separated by semicolons if more than one is needed

- You can control the **background**, **foreground** and **bold** settings in your terminal

- **`Note`**`: You will need to reset the colour codes at the end of your prompt to avoid the colour being applied to typed text. The numeric code 0 is used to reset the background, foreground and bold settings, as follows: \e[0m`

PTA Linux Series

Shell– Colouring Customisation Prompts

- The following chart can be used to choose your colour and bold settings:

Source: IBM Developer Works: www-128.ibm.com/developerworks/linux/library/l-tip-prompt

Shell Customisation - Colour Prompts

- The rows indicate text colour, while the columns indicate background colour
- For example, to set your text to be white on a blue background, we need numeric colour codes: 37 for white text and 44 for blue background (**\e[37;44m**). We will apply these settings to the current user, time, and directory with a dollar sign for the end of our prompt (**\u\t\w$**)

```
export PS1="\e[37;44m\u\t\w$"
```

- Unless we want to keep this colour scheme for our typed in text, we will also need to reset our terminal **(\e[0m)**

```
export PS1="\e[37;44m\u\t\w$ \e[0m"
```

PTA Linux Series

Shell Customisation – Finishing Touches to Our Colour Prompt

- In order to make the text more readable, we can set the bold face by adding a 1 as follows:

```
export PS1="\e[37;1;44m\u\t\w$ \e[0m"
```

- **Note**: When we customised our prompt **word-wrapping** was turned off, in order to fix this we need to escape all non-printable characters so that long commands of text are wrapped onto the next line again. The bash escape sequences for this are \[and \]

```
export PS1="\[\e[37;1;44m\]\u\t\w$ \[\e[0m\]"
```

PTA Linux Series

Exercises

1) Change into your **project1** directory inside your home directory, re-edit your **report.txt** file using the least amount of keystrokes possible
2) Check to see which directories are searched for programs that you type in
3) Create a shorter alias for emacs
4) Create a file called **shopping_list** with the following 3 lines:

```
bread, butter, tea, biscuits
apples, 6 oranges, strawberries
chocolate, ice-cream, 1 cake
```

- Now, search for a line beginning with “apples”
- Search for any line which contains the letters “ea”
- Search for a line ending with “s”

PTA Linux Series

Exercises

5) Save the output of **`ls`** to a log file for later use

6) Customise your terminal to have green text on a black background with the current time and current directory as the prompt

7) Edit your bash configuration file in your home directory to save your settings for your customised prompt

8) Edit your $PATH shell variable so that when when you issue a command from the prompt, the shell goes to look for a program of that name by *first looking* in your current directory. This means modifying the shell variable called $PATH so that it starts with '.:'. The colon is used as a separator and the dot is shorthand for “current directory.” First you will need to find what shell config file the $PATH variable is defined in!

   This exercise is useful because it means that instead of having to type '`./myprog`' to run the program of that name in the current directory you can now simply type '`myprog`'. Doesn't seem like much now but wait until your back is to the wall and you're tearing your hair out trying to get an assignment working...

PTA Linux Series

Lab02 – Part III

Files and the File System

Professional Training Academy Linux Series

PTA Linux Series

File Ownership

- There are 3 types of permissions in Linux:
  - **Read**: r
  - **Write**: w
  - **Execute**: x

- There are also permissions for 3 different types of users:
  - the file **owner** (u)
  - the file owner's **group** (g)
  - **other** users (o)

PTA Linux Series

File Ownership

- The long listing argument to the `ls` command will show the file ownership details, for example:

```
ls –l filename
-rwxr-xr-x 1 ict 0 Jan 1 12:00 filename
```

- The first letter indicates a **directory** (d) or a **file** (-)
- The following 3 letters (rwx) are the permissions of the **owner**
- The next 3 letters (r-x) are the **group** permissions
- The final 3 letters (r-x) are the permissions for **other** users
- **Note**: If a flag is not set, the dash sign is used **-**

PTA Linux Series

Changing the Owner

- In the following example, the ict user owns the file:

```
ls –l filename
-rwxr-xr-x 1 ict 0 Jan 1 12:00 filename
```

- The change ownership (**chown**) command can be used to change the owner of a file

- To change the ownership over to the csis user, type:

```
chown csis filename
```

PTA Linux Series

Changing the Permissions

- In the following example, other users can read the file

```
ls –l filename
-rwxr-xr-x 1 ict 0 Jan 1 12:00 filename
```

- The change modification (**chmod**) command can be used to alter permissions using a plus sign to add or a minus sign to remove a permission. For example, to take away the read permission for other users, type:

```
chmod o-r filename
```

- Likewise, to add write permissions to the filename, type the following:

```
chmod g+w filename
```


Hidden Files

- Hidden files in UNIX and Linux begin with a **dot**. For example, we have already seen the `.profile` configuration file

- Hidden files are generally configuration files and places to save program settings

- To view hidden files, we can use an argument with `ls`, as follows (where `a` means list **all files**):

```
ls -a
```

- We can also use the `wildcard` special character to list all hidden files, as follows:

```
ls -a .*
```

PTA Linux Series

Comparing Files

- To compare the contents of two or more files, you can use the **`diff`** command

- The `diff` command outputs the **differences** between the compared files

- For example, to compare two files named file1 and file2, type the following:

```
diff file1 file2
```

- **`Note:`** `Diff will print the lines which differ between the two files`

PTA Linux Series

Creating Symbolic Links

- Symbolic links are equivalent to **shortcuts** on Windows
- You can create a sym link in 2 different locations between:
  - programs
  - directories
  - files, i.e. configuration files
- Example: you have installed a new program in your home directory called `coffeemaker` and you want to make it available to all other users on your system
- For this, we will need to use the **`ln`** command to create a symbolic link to our new location:

```
ln –s /usr/bin/coffeemaker ~/coffeemaker
```


Creating Archives

- There are many ways of archiving in Linux
- You can use **`zip`**, and these will also open on Windows and vice-versa
- You can also use **`tar`** or **`bunzip`** or even **`jar`**
- These can all be applied in the same way
- That is, you have a large file or a folder full of files/folders which need to be compressed into a smaller space on disk

Creating Zip Files

- If you have a folder full of files called `myProject`, you can create a zip file as follows:

```
zip –r myProject.zip myProject/*
```

- The **`–r`** means recursive. In other words, zip up everything inside our folder
- The first option given to zip must be the name of our newly compressed file
- We can then tell zip which directory to compress
- You can also list a set of files instead of directory

Unpacking Zip Files

- To unpack or unzip our file we do the following:

```
unzip myProject.zip
```

- The zip program will then unpack all the files into your current working directory

- **Note**: you should create regular backups for your files at work or in college. You can burn these on CD or copy them to a USB Key

Creating Tar Files

- To create a tar file of our `myProject` directory, we can do following:

```
tar –cvf myProject.tar myProject
```

- The `c` means **create**
- The `v` means **verbose** (print out info.)
- The `f` means our **filename** is `myProject.tar`
- As with zip, you can also list a set of files instead of the directory as the last argument
- **`Note`**`: The same arguments (cvf) are also used when creating jar files`

Unpacking Tar Files

- To unpack our file, do the following:

```
tar –xvf myProject.tar
```

- The `x` means **extract**
- Again, `v` means **verbose** output
- And `f` means **file** again
- Excellent talking point at your next cocktail party: did you know that 'tar' is a contraction of 'tape archive' (as in magnetic tape), which is how the archives were stored fadó, fadó.

Creating Log Files

- Previously, we learned about the redirection pipe operator. If you need to redirect the output of a program to an existing file you can use the pipe operator. For log files, however, it is more useful to use the greater than operator, as it will create the file for you if it does not exist

- For example, if you want to record the output from the `ls -a` command, type:

```
ls -a > myLogFile
```

- The output of the `ls` command will be saved in a file call `myLogFile`

Creating Log Files

- If `myLogFile` does **not exist**, it will be created
- If `myLogFile` does **exist**, it will be overwritten
- To append to the end of a file instead, use the double arrow operator instead of the single arrow operator, as follows:

```
ls -l >> myLogFile
```


System Log Files

- The log files for the Linux system are stored in the `/var/log` directory
- For example, any problems or error messages from the system are stored in the `messages` log file
- We can use the `less` command to view this file as normal

```
less /var/log/messages
```

PTA Linux Series

System Log Files

- The **`less`** command is difficult to use with very large files such as these, especially when we just want to see the most recent entry appended to the end of the file

- Another command called **`tail`** is more useful as it prints the last 10 lines of a file

- For example, to print the last 10 lines of the messages log file:

```
tail /var/log/messages
```

- You can also specify how many lines to print, for example, to print the last 50 lines, use the number argument **n:**

```
tail –n 50 /var/log/messages
```


System Log Files

- There is another argument for tail which will keep updating the output as the busy log files changes
- This is **`tail`** with the **`-f`** argument
- To keep an eye on the latest system messages appended to the `/var/log/messages` file, use the following:

```
tail –f /var/log/messages
```


Installing Files

- Installing files on Linux requires you to be aware of how the program is packaged
- For example, is the program a zipped up C++ project which you must compile?
- Or has it already been compiled for a system like yours and saved as an **`rpm`** file?
- Or can you use **`apt`** to update it?

PTA Linux Series

Installing from Source

- If you have to compile a C or a C++ program from source code, you will need the following 3 steps

- **Note:** You need to be root for the last step, as it installs the program in a system directory for everyone to use

```
./configure
make
make-install
```


Installing an RPM

- The Red Hat Package Manager is a common format for distributing files on Linux, and makes installation much easier

- For example, if you have a program called `mygame.rpm`, install as follows:

```
rpm –ivh mygame.rpm
```

- The `i` means **install**

- The `v` means **verbose**

- The `h` means display the installation progress as a series of **hashes**

Using Apt

- Apt – Debian Automatic Package Management
- Configure your sources in `/etc/sources.list`
- To sync your local list with the online repository, type the following:

```
apt-get update
```

- Now you are ready to install or update your programs

Using Apt

- For example, to **install** the Mozilla Firefox internet browser:

```
apt-get install mozilla-firefox
```

- To **update**:

```
apt-get upgrade mozilla-firefox
```

- To **un-install**:

```
apt-get remove mozilla-firefox
```

PTA Linux Series

Installation Notes

- When installing from source or manually installing rpm files, the program you wish to install will very often have dependencies
- This means that the program depends on other programs or libraries to be installed for its use
- These dependencies need to be installed on your system before installing your program
- However, life is made easier when we use **`apt`**, as apt will automatically download dependencies for us

PTA Linux Series

Open Source Repositories

- Now that you know how to install Linux programs, the following two sites will provide you with an excellent resource for programs

- www.sourceforge.net

- www.freshmeat.net

PTA Linux Series

Exercises

1) Find out what time you created the **report.txt** file
2) Change the permissions on your **report.txt** file so that nobody else but you can read the report
3) Make a copy of **report.txt** called **manager_report.txt** and add a new line. In the terminal, find out which line is different between the two
4) Create a symbolic link from **manager_report.txt** to a file in your home directory called **latest_report.txt**
5) Create a zipped folder called **meeting_files.zip** from the **project1** directory
6) Open up you system log file so that it updates in real time