Birzeit University - Faculty of Engineering & Technology
Electrical & Computer Engineering Department - ENCS313
Linux laboratory

Experiment #1
Getting started with Linux

0.1 Introduction

The experiment intends to have students started on the Linux OS and discover some of its features. Basic commands and environment variables will be shown as well as file and directory permissions.

0.2 Objectives

The objectives of the experiment is to learn the following:

- Log in/out your Linux OS with your username/password.
- Start a command window and type some basic commands: directory navigation and control, file manipulation commands, command options, command aliasing, display commands.
- Environment variables.
- Understand the file/directory permissions.

0.2.1 Command windows

1. Login in your Linux account using your login name and password.
2. Start a command window by executing the following menu sequence:
   `Application` → `System` → `Terminal` (assuming your Linux distribution is Red Hat Fedora). If your Linux distribution is Ubuntu, execute the following menu sequence:
   `Application` → `Accessories` → `Terminal`.
3. Execute the command `pwd (print name of current/working directory)` and make sure you are in your home directory.
4. Execute the command `cd (change directory)` and hit `Enter`. Notice that your location remains unchanged.
5. Execute the command `cd ..` followed by the command `pwd`. Notice the current location you are in.
6. Execute the command `cd` and hit `Enter`. Notice that your location is back to that of step **3**. Conclusion: whenever you type command `cd` and hit `Enter`, your location becomes that of the home directory.
7. If you execute the command `history`, you should be able to see the history of commands you previously typed preceded by a serial number. To re-execute a command, you can thus look it up in the history list. Once you locate it, precede the command by an exclamation sign followed by the related number.

   Example:

   Assume a partial ouput of the command `history` was as follows:

```
82 pwd
```

To re-execute the command `pwd`, you can execute the command ! 82 instead of executing the command `pwd`.

8. Once you type a command on the command window, you can navigate on that command as follows:
   - `Ctrl-a`: move the cursor to the beginning of the command line.
   - `Ctrl-e`: move the cursor to the end of the command line.
   - `Ctrl-f`: move the cursor one character forward on the command line.
   - `Ctrl-b`: move the cursor one character backward on the command line.
   - `Esc-f`: move the cursor one word forward on the command line.
   - `Esc-b`: move the cursor one word backward on the command line.

   In addition, the following commands might be useful as well:
   - `Ctrl-t`: transpose 2 characters. For example, you can type on the screen the string:

     `teh`

     If you wish to transpose the letters `e` and `h`, move the cursor over the letter `h` and click `Ctrl-t`. You will notice that the string becomes displayed as follows:

     `the`
   - `Esc-t`: transpose 2 words. For example, you can type on the screen the string:

     `encs313 ls`

     If you wish to transpose the words `encs313` and `ls`, move the cursor to the blank position between the 2 words and click `Esc-t`. You will notice that the phrase becomes displayed as follows:

     `ls encs313`
   - `Ctrl-l`: clear the command window and move the cursor to its initial top position. This is a shortcut for the command `clear` but much faster to type.
   - `Esc-u`: Change word to *upper* case.
   - `Esc-l`: Change word to *lower* case.
   - `Esc-c`: Capitalize word.

9. To repeat previously typed commands, use the up arrow of your keyboard (↑). Everytime you click on the ↑, an older command will show up. Hitting `Enter` will execute that command.

10. Execute the command `exit`. Your command window will be closed. You get the same behavior if you execute `Ctrl-d`.

The command window we've seen in this section will be your main interface with the system. In that window, you'll execute commands and the output will be displayed. Note that you can open as many command windows as you need.

0.2.2 Directory navigation and Control

1. Start a command window.

2. Execute the command `ls (list directory contents)` and check the content of the folder you're at. Notice that executing the `dir` command gives the same result.

3. Execute the command `mkdir encs313` followed by the command `ls`. Notice that the directory `encs313` has been created. You can move to the newly created directory by executing the command `cd encs313`.

4. Execute the command `cd .` or `cd ./` followed by the command `pwd`. Notice that your current location remains unchanged. The `./` or simply `.` refer thus to the current location.

5. Execute the command `cd ..` or `cd ../` followed by the command `pwd`. Notice that your current location has moved one step up. The `../` or simply `..` refer thus to the parent directory.

6. Execute the command `cd -` followed by the command `pwd`. Note the current dierctory you're at. Execute the command `cd -` followed by the command `pwd` again and note the current dierctory you're at. You will notice that the command `cd -` gets you always to the location of the previous directory you were at. That command is quite useful whenever you keep on jumping between 2 directories. Linux doesn't define a `cd +`. Can you explain why?

7. Execute the command `cd` and hit `Enter`. Your current location should be the home directory. Execute the command `rmdir encs313` followed by the command `ls`. Notice that the directory encs313 you created previously has been removed.

Note:

The path that starts with a `./` or `../` are called relative paths since they are referenced with respect to the current location. Executed commands that involve the usage of relative paths might give different results if the current location is different.

The paths that start with a `/` are called absolute paths since they relate to the `root` folder. Executed commands that involve the usage of absolute paths give always the same result irrelevant of the current location.

Example:

`cd ..` means move to the parent directory relative to the current location.

`cd /home/user` means move to the `user` directory no matter where your current location is.

0.2.3 File manipulation commands

1. In a command window, go to the home directory and create the directory `encs313`. Move to the newly created directory by executing `cd encs313`.

2. Execute the command `touch myFile.txt` followed by the command `ls`. Notice that a file called `myFile.txt` has been created. The file size is initially 0 since it is empty.

3. To copy the file `myFile.txt`, execute the command:

   ```
   cp myFile.txt myFile1.txt
   ```

4. Execute the command `ls` and notice that you have 2 files: `myFile.txt` and `myFile1.txt`.

5. To delete the newly created file, execute the command:

   ```
   rm myFile1.txt
   ```

6. Execute the command `ls` again and notice that the file `myFile1.txt` has been removed.

7. To rename a file, execute the command:

   ```
   mv myFile.txt myFile1.txt
   ```

8. Execute the command `ls` and notice that the file `myFile.txt` has indeed been renamed to `myFile1.txt`. Note that you can use the command `rename myFile.txt myFile1.txt` and obtain the same result. However, the command `mv` is more popular.

9. Beside being used to rename files, the command `mv` is used to move files and directories around. For example, if you intend to move the file `myFile.txt` to the parent directory, you can execute:

   ```
   mv myFile.txt ../
   ```

   To move the file `myFile.txt` back under directory `encs313`, execute:

   ```
   mv ../myFile.txt .
   ```

   Note that since directory `encs313` exists, the file `mv myFile.txt` will NOT be renamed into `encs313` but moved under `encs313`.

0.2.4 Command options

The power of Linux doesn't come only from the rich set of commands it has but also from the different options that these commands take as argument. The output display of a specific command will be formatted differently if supplied with different options. Let's consider the following steps and check the output at each case:

1. Execute the command `cd` and hit `Enter`. Your current location should be the home directory. Execute the command `ls` and note the output you get.
2. Execute the command `ls -l` and note the output you get.
3. Execute the command `ls -a`, `ls -aF` and `ls -r` respectively and note the output you get at each execution. What would you conclude when comparing the different outputs?
4. Execute the command `man ls` and check the list of options that the command `ls` might take as argument. You would notice a list of 30+ options for the command `ls` alone. Other Linux commands might even take more options!

   Note:

   The `man` command invokes the manual pages on the argument you supply. It is sort of help that you get regarding that particular command. If the supplied command doesn't exist, an error message will be displayed in the command window. Otherwise, the command help page is invoked. In general, most of the Linux commands have associated manual pages.

   To get out of a manual page, click on the character `q` (quit).

We'll show below an additional example on the options that the command `rm` might take:

a. Execute the command `cd` and hit `Enter`. Your current location should be the home directory.
b. Execute the command `mkdir encs313`.
c. Execute the command `cd encs313`.
d. Under directory `encs313`, create an empty file by executing `touch myFile.txt`.
e. Go back to the parent directory and execute the command `rmdir encs313`. What do you get?
f. Execute the command `man rm` and note the different options that you get. You'll note in particular the option `-r` that removes directories and their contents recursively.
g. Execute the following command:

   ```
   rm -r encs313
   ```

h. Note the result you obtain in the previous step.

To do:

- Check the option `-i` of the command `rm`. Try it and explain the behavior you get.
- Check the different options for the following commands and try to get familiar with them: `ls`, `cp` and `mv`.

0.2.5 Command aliasing

The behavior of a specific command can be altered to that of one of its options by using command aliasing. For example, assume that you always need to run the command `ls` with the 2 options `-a` and `-F`. You thus need to execute:

```
ls -aF
```

To make the command `ls` behave as `ls -aF`, you can do the following:

```
alias ls='ls -aF'
```

Note that you are not allowed to have any blanks before or after the `=` sign. The command `ls` will from now on behave exactly as `ls -aF`.

Note the following:

- If at any point you need to run the original version of the `ls` command, you can execute:
  `\ls`

Note that the alias you just created will have effect in the command window where it has been defined. To make the alias visible to all future command windows, check the following sections.

To remove an alias on a particular command, you can use the command `unalias`. For example, you can remove the alias on the command `ls` by executing:

```
unalias ls
```

0.2.6 Display commands

There are a number of commands you can use to **display** or **view** a file. Some of these are editors which we will look at later. Here we will illustrate some of the commands normally used to display a file.

`echo` command

You can use the command `echo` to display a string on the standard output as follows:

```
echo ``Hello, World!''
```

The output will display:

Hello, World!

As a second example:

```
echo ``1 + 2 + 3 + 4''
```

The output will display:

1 + 2 + 3 + 4

Note that the use of double quotations are optional. You get the same output without quotations as well.

Note as well as the command `echo` can take many arguments that affect the way it display strings. Execute `man echo` to see the different options.

`cat` command

You can use the command `cat` to concatenate a file or list of files (check on the different options of command `cat` by executing `man cat`). To display the content of file `myFile.txt` on the

standard output, you can execute:

```
cat myFile.txt
```

If you have multiple files and you want to display their content, you can execute:

```
cat myFile.txt myFile1.txt
```

Note that the content of the file `myFile.txt` will get displayed before the content of file `myFile1.txt`.

An interesting option that can be used with the command `cat` is the `-n`. For example, if you execute:

```
cat -n myFile.txt
```

the line number will be displayed infront of each line.

`more, less` commands

The `more` command behaves somehow like the command `cat`. It's behavior differs though between the different Linux distributions. On some of them, it displays one page at a time while on others it displays the whole content of the file (e.g. behaves like the command `cat`). As an example, execute the following command:

```
more myFile.txt
```

The commands `less` and `more` display one page at a time. Their behavior can be summarized as follows:

- Hiting on the space bar moves the displayed file one page down.
- Hiting `Enter` moves the displayed file one line down.
- Clicking on the letter `p` moves the displayed file to the first page.
- Clicking on the letter `b` moves the displayed file one page up.
- Clicking on the letter `q` quits the command.

`head, tail` commands

The command `head` displays the first 10 lines of a file while the command `tail` displays the last 10 lines.
As an example, execute and note the output of the following 2 commands:

```
head myFile.txt
tail myFile.txt
```

The behavior of these commands can be altered if supplied by an argument. For example, if you execute the following command:

```
head -1 myFile.txt
```

the file's first line only will be shown on the standard output. The last line of a file gets displayed if you execute:

```
tail -1 myFile.txt
```

0.3 Environment variables

Environmental variables are used to provide information to the programs you use. You can have both **global environment** and **local shell variables**. Global environment variables are set by your login shell and new programs and shells inherit the environment of their parent shell.

Local shell variables are used only by that shell and are not passed on to other processes. A child process cannot pass a variable back to its parent process[1].

The current environment variables are displayed with the `env` or `printenv` commands. Some common ones are:

DISPLAY	The graphical display to use, e.g. nyssa:0.0
EDITOR	The path to your default editor, e.g. /usr/bin/vi
HOME	Path to your home directory, e.g. /home/frank
HOST	The hostname of your system, e.g. nyssa
LOGNAME	The name you login with, e.g. frank
PATH	Paths to be searched for commands, e.g. /usr/bin:/usr/ucb:/usr/local/bin
SHELL	The login shell youre using, e.g. /usr/bin/csh
TERM	Your terminal type, e.g. xterm
USER	Your username, e.g. frank

Many environment variables will be set automatically when you login. You can modify them or define others with entries in your startup files or at anytime within the shell. Some variables you might want to change are `PATH` and `DISPLAY`. The `PATH` variable specifies the directories to be automatically searched for the command you specify.

Setting of environment variables depend on the shell you are using. Most probably your shell is the `bash` shell. To set an environment variable, do the following,

1. Execute the command `env` and make sure your shell is the `bash` shell.
2. If your shell is `bash`, set a dummy environment variable `DUMMY` to the value `ENV_VALUE` as follows:

   ```
   DUMMY=ENV_VALUE
   ```

   Note that you should leave no blanks before and after the = sign. Note as well that different shells might set the environment variables in different ways. For example, if your shell is `tcsh`, then setting an environment variable becomes as follows:

   ```
   setenv DUMMY ENV_VALUE
   ```
3. To get the value of an environment variable, do the following:

   ```
   echo $DUMMY
   ```
4. Environment variables can be used with Linux commands. An example would be as follows:

   ```
   cd $HOME
   ```

The environment variable `$DUMMY` that you just defined above has a scope limited to the command window where it was defined. This is the exact case that you encountered when you defined aliases in the previous sections. To define environment variables or aliases and make them visible in all command windows, you need to define them in a special hidden file. The name of that hidden file depends on the type of the shell you are using:

- If your shell is **sh** (Bourne shell), your hidden file's name is `.profile`.
- If your shell is **bash**, your hidden file's name is `.bashrc`.
- If your shell is **csh**, your hidden file's name is `.cshrc`.
- If your shell is **tcsh**, your hidden file's name is `.tcshrc`.

[1]Extracted from the book entitled: **Introduction to Unix** by Frank G. Fiamingo, Linda DeBula, Linda Condron - University Technology Services - The Ohio State University.

These hidden files must be located under your home directory. To know the type of shell you are using, you need to check the environment variable `$SHELL`.

As an example, consider you want to define an environment variable called `DUMMY` and set an alias for the command `ls` to `ls -aF`, you need to do the following (assuming your shell is `bash`):

- Open the hidden file `.bashrc` that is located under your home directory.
- Add the following 2 rows:

  ```
  DUMMY=ENV_VALUE
  alias ls='ls -aF'
  ```

- Save the file `.bashrc` and close it.
- Open a new command window and check that the environment variable `DUMMY` is defined (`echo $DUMMY`) and that the alias on the command `ls` is defined (`alias ls`).

0.3.1 File permissions

By file permissions, we mean the permissions on files and directories alike. Each file, directory, and executable has permissions set for who can **read**, **write**, and/or **execute** it. To find the permissions assigned to a file or a directory, the `ls -l` command should be used. The following might be obtained:

```
> drwxr-xr-x   3 root    other        512   Apr 22  1999 myDir
> lrwxrwxrwx   1 root    other          2   Nov  4  12:29 xq -> xt/
> -rw-r--r--   1 root    other       1338   Apr  7  2005 myFile.c
```

Before explaining the letters shown above, in particular under the first column, it is important to note that Linux has 3 levels of permissions:

- Read, write and execute permissions for the owner (`rwx`).
- Read, write and execute permissions for the group to which the owner belongs (`rwx`).
- Read, write and execute permissions for others (`rwx`).

Put together, that makes the following permissions for the owner, the group and others:
`rwxrwxrwx`.

The first letter varies according to the type of the file:

- If the letter is `d`, the file is a directory.
- If the letter is `l`, the file is a symbolic link.
- If the letter is `-`, the file is a regular file.

In summary, we can state the following:

- `myDir` is a directory, read/write/execute permissions are granted to owner, read/execute permissions are granted to the owners group, read/execute permissions are granted to others.
- `xq` is a symbolic link (as can be seen in `xq -> xt`. Read/write/execute permissions are granted to owner, group and others.
- `myFile.c` is a file. Read/write permissions are granted to owner while only read permissions are granted to group and others.

To set permissions on files/directories, you can use the command `chmod`. For example, to change the permissions on file `myFile.c` to read/write/execute for the owner only, no permissions for the group, no permissions for others, you can execute the following:

`chmod 700 myFile.c`

Alternatively, to change the permissions to read/write/execute for the owner, read/execute permissions for the group and read permission only for others, you can execute:

`chmod 754 myFile.c`

An alternative way for setting the permissions would be as follows:

`chmod u=rwx,g=,o= myFile.c` (same as `chmod 700 myFile.c`)

`chmod u=rwx,g=rx,o=r myFile.c` (same as `chmod 754 myFile.c`)

0.3.2 Links to files

The `ln` command creates a "link" or an additional way to access (or gives an additional name to) another file. If the link is symbolic, the link is a shortcut that points to a file or a directory. To create a symbolic link, execute:

`ln -s myFile.c link`

In the above command, `link` becomes a shortcut for the file `myFile.c`. You can open the file `myFile.c` by opening `link` and close the file `myFile.c` by closing `link`. Note though that removing `link` will not remove the file `myFile.c`. The shortcut itself is removed only.

To do:

Create a link to your home directory and make sure that your location is that of the home directory once you `cd` using that link.

A **hard link** can only be done to another file on the same file system, but not to a directory (except by the superuser). A hard link creates a new directory entry pointing to the same inode as the original file. The file linked to must exist before the hard link can be created. To link two files with a hard link to each other execute:

`ln sourceFile destinationFile`.

Note that if you remove file `sourceFile`, file `destinationFile` will remain.

0.3.3 Appending files - redirection

We've seen in the previous experiment that with command `echo` we can output strings to the standard output. In this section, we'll use that command in addition to redirection to append lines of data to a file without the need to open files explicitly.

a. `cd` to your home directory.

b. Create a new file called `newFile.txt` using the command `touch`.

c. Execute the following command:
`echo "Hello, World!" >> newFile.txt`

d. Execute the command `cat newFile.txt`. Note the content of the file `newFile.txt`.

e. Repeat step **(c)** multiple times then note the content of the file `newFile.txt`.

We say that the symbol `>>` redirects the output from the standard output to the file.

Note:

If you execute the command `echo "Hello, World!" > newFile.txt`, the content of the file `newFile.txt` will be overwritten! Thus, be aware to differentiate between symbols `>>` and `>`.

The redirection examples we've seen above related to **output redirection**. Just as the output of a command can be redirected to a file, so can the input of a command be redirected from a file. And as the greater-than character > is used for output redirection, the less-than character < is used to redirect the input of a command. Of course, only commands that normally take their input from standard input can have their input redirected from a file in this manner.

Example

- Create the file `redir.txt`
- Execute the following 2 commands:

  ```
  echo "Line 1" > redir.txt
  echo "Line 2" >> redir.txt
  ```

- Execute the following command:

  ```
  cat < redir.txt
  ```

- You get the same result as in the previous step by executing:

  ```
  cat redir.txt
  ```

Note that in the above example with input redirection, the command `cat` did not know that it was reading from the file `redir.txt`; it only knows that it is reading its input from standard input.

Error redirection

In addition to standard input and standard output, there is another place known as **standard error**. This is where most Unix commands write their error messages. And as with the other two "standard" places, standard error is associated with your terminal by default. In most cases, you never know the difference between standard output and standard error.

Example

```
ls zzz*
```

You get the following error:

```
zzz* not found
```

Here the `"not found"` message is actually being written to standard error and not standard output by the `ls` command. You can verify that this message is not being written to standard output by redirecting the `ls` command's output:

```
ls zzz* > foo
```

You still get the error message:

```
zzz* not found
```

Thus, error messages will still get displayed at the terminal even if standard output is redirected to a file or piped to another command.

You can redirect standard error to a file by using the notation:

```
ls zzz* 2> errors
```

No space is permitted between the `2` and the `>`. Any error messages normally intended for standard error will be diverted into the specified file (in the above example, the file `errors`), similar to the way standard output gets redirected. If you execute the command:

```
cat errors
```

You'll get the output:

```
zzz* not found
```