Introduction to Linux Command Line Interface

Fadjar Fathurrahman

Department of Engineering Physics
Research Center for Nanoscience and Nanotechnology
Bandung Institut of Technology

Overview

This talk is intended to teach the basics of working with command line interface (CLI) of Linux for those who never used CLI before.

From Wikipedia:

> A command-line interface (CLI) is a means of interacting with a computer program where the user (or client) issues commands to the program in the form of successive lines of text (command lines). The interface is usually implemented with a command line **shell**

We will limit our discussion to **bash** shell because this is the most popular shell in Linux.

Command line interface

Advantages:

- Requires fewer resources
- Concise and powerful
- Expert-friendly
- Easier to automate via scripting
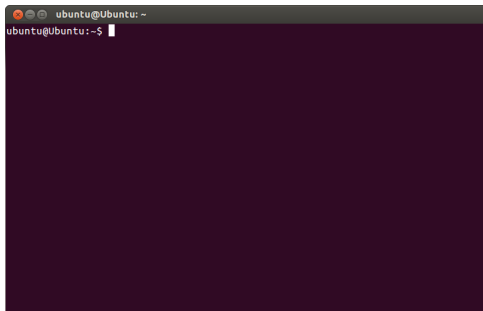
Disadvantages

- Unintuitive
- Commands not obvious
- Not visually rich
- Beginner-unfriendly

Accessing command line interface

There are at least two ways of doing this:

- Via menu: Accessories Terminal
- Using special shortcut: **Ctrl + Alt + T** (on Ubuntu)

Displaying text on screen

Type the following (and press enter at the end of line)

```
echo "My name is XXX"
echo My name is XXX
```

Both will result in the same output.

I personally prefer the first style.

Working with variables

Built-in variable USER

```
echo $USER
# to access the variable we append a dollar sign in front
# of the variable name

echo USER   # notice that there is no dollar sign

echo "Hello, I am logged in as $USER"
```

Define new variable: (Note that there is no space after variable name)

```
MYNAME="Fadjar Fathurrahman"
echo "Hello, my name is $MYNAME"
```

Be careful with space(s)

```
MYNAME=Fadjar Fathurrahman # will result in error
```

Working with variables (cont'd)

Define a variable named `AWESOME_ACTORS`:

```
AWESOME_ACTORS="Johnny Depp, Morgan Freeman"
echo $AWESOME_ACTORS
```

Add some actors to `AWESOME_ACTORS`:

```
AWESOME_ACTORS="Samuel L Jackson, Robert Downey Jr., $AWESOME_ACTORS"
echo $AWESOME_ACTORS
```

Redefine `AWESOME_ACTORS`

```
AWESOME_ACTORS="Christian Bale, Matt Damon"
echo $AWESOME_ACTORS
```

Working with variables (cont'd)

Other useful built-in variables:

```
echo $PATH
echo $LD_LIBRARY_PATH
echo $HOME
```

PATH tells the shell which directories to search for executable files (i.e. ready to run programs) in response to commands issued by a user.

LD_LIBRARY_PATH is similar to PATH, but applies for libraries.

HOMEis location of user's home directory, usually /home/username. It can be shortened using ~.

Files and directories

```
pwd  # display current directory
ls   # display list of files in current directory
```

Go to specific directory (changing directory)

```
cd SecretDirectory   # go to a directory with name SecretDirectory
cd ../               # go "up"
cd $HOME             # go to home directory
cd                   # same as cd £HOME
cd /                 # go to directory /
```

Create new directory

```
mkdir MyNewSecretDirectory
```

Files and directories (cont'd)

Be careful with spaces: (again)

```
mkdir My New Secret Directory # will create several directories
mkdir "My New Secret Directory"
cd My New Secret Directory    # will go to the first directory, i.e. My
cd "My New Secret Directory"
```

To delete files we can use the command `rm`. To delete a directory we can use `rm -r`

```
rm MySecretFile.txt       # delete a file
rm -r MySecretDirectory   # delete a directory (and all files in it)
```

`rm` command is considered VERY DANGEROUS. Be careful when using it.

Files and directories (cont'd)

To **rename** a directory or a file we can use the command `mv`

```
mv DirName BetterDirName
mv OldName.txt NewName.txt
```

Command `mv` also can be used to **move files and directories** to another directory:

```
mv file1.txt file1.txt dir2 dir2 DestinationDir
```

Files and directories (cont'd)

The cp command can be used to **copy** files.

Copy file1 to file2

```
cp file1 file2
```

Copy two files from upper directory to current directory ('./' or simply '.')

```
cp ../file1 ../file2 ./
```

Copy directory dirname to current directory

```
cp -r ~/dirname/ ./
```

Files and directories (cont'd)

The command `ls` has several options:

```
ls
ls -l
ls -al
ls -alh
```

List all files in a directory named `MySecretDirectory`:

```
ls MySecretDirectory
ls -l MySecretDirectory
ls -al MySecretDirectory
ls -alh MySecretDirectory
```

Text editor(s)

Create empty file:

```
touch A_Whole_New_File.txt
```

Edit the file (use either nano or gedit)

```
nano A_Whole_New_File.txt
gedit A_Whole_New_File.txt &
```

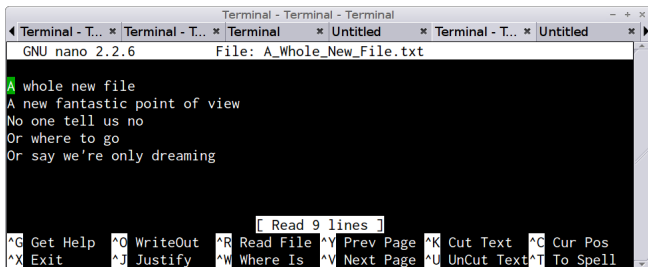Type the following (or other text if you wish)

```
A whole new file
A new fantastic point of view
No one to tell us no
Or where to go
Or say we're only dreaming
```

Text editors (cont'd)

nano is more difficult to use as compared to gedit.

'gedit' is very much like Notepad in Windows.

Text editors (cont'd)

less can be used if we only interested in viewing the content of a text file. For example try:

```
less A_Whole_New_File.txt
```

To quit from less type q.

There are a lot of text editors on Linux:

- ▶ vim (http://www.vim.org)
- ▶ emacs (www.gnu.org/software/emacs)
- ▶ atom (http://atom.io)
- ▶ See the list on Wikipedia https://en.wikipedia.org/wiki/Category:Linux_text_editors

Interesting read: https://en.wikipedia.org/wiki/Editor_war

Redirection and pipelining

Sometimes, we want to save what displayed after in the terminal after calling a command/program. We can do this by **redirecting** the output to another file (usually a text file).

```
ls -lh > file_list.txt
less file_list.txt
```

If we want the output to be displayed in the terminal and also written in a file, we can use tee.

```
ls -lh | tee file_list.txt
```

Note that in the previous example, we also have used **pipelining** by using the character |. This will take the output of the first command, i.e. `ls -lh` and supplied it to tee which will display and write to a file.

Input and output file

In the following workshops, we will work with various programs.

These programs usually need to read an input file.

The input file can be provided by redirection using symbol < (in this case, the program read the input file from standard input)

```
AProgram < input_file
```

Other programs may need to be supplied input file name as an argument to the program

```
AnotherProgram input_file
```

Input and output file (cont'd)

In most cases, the program will give some messages or important output on the standard output (screen).

If this is the case, the we may use the following to redirect the output to a file:

```
AProgram < input_file > log_file
AnotherProgram input_file > another_log_file
```