CENG328 Operating Systems Laboratory Chapter 1

Linux is the common name given to a large family of operating systems. All Linux-based operating systems are essentially a large set of computer software that are bound together. The most essential piece of these operating systems is the “Linux kernel”. A large variety of software may be installed and run on this kernel, including web browsers, text editors, multimedia players, games; or more specialized software such as scientific tools, web and database servers, etc.

As of 2018, there are over 100 different Linux-based operating systems. Ubuntu, Mint, Debian, OpenSUSE, Fedora are only a few of the most popular ones. The differences between all these operating systems are usually in terms of the user interface or how the operating system handles the background tasks. For example Kubuntu is different from Ubuntu only in terms of user interface and the default set of programs bundled with them; but all the other parts (Linux kernel, system services, etc) are exactly the same and common operations can be performed in a similar manner in both operating systems.

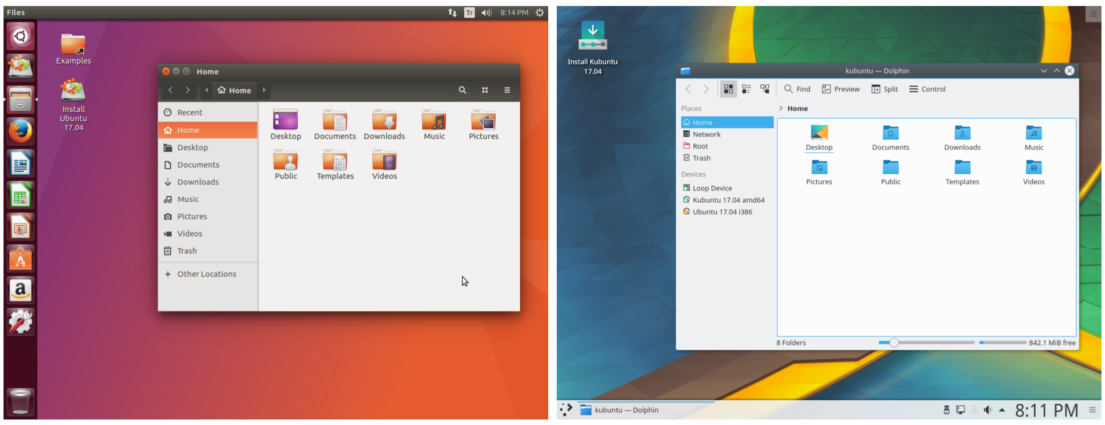


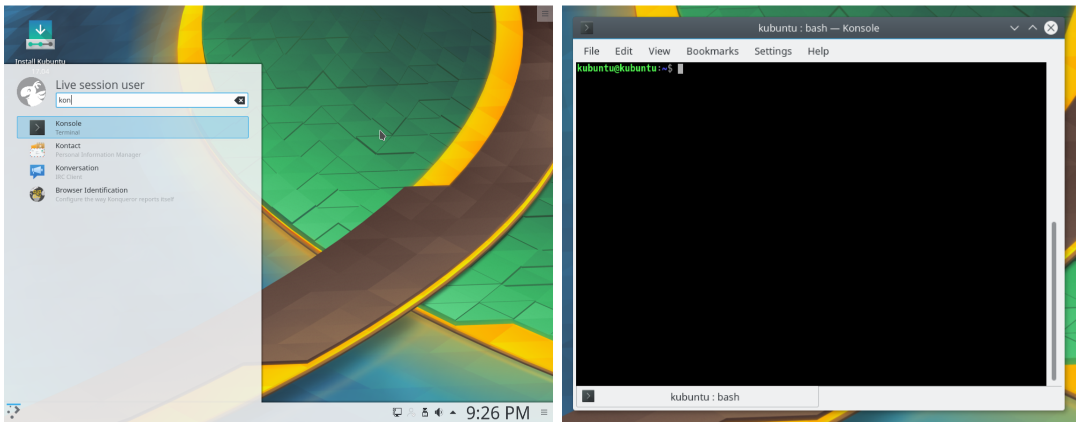
Figure 1: Ubuntu and Kubuntu desktops

We will be using Kubuntu operating system in our university laboratories but you are free to install any Linux-based operating system on your own computers. Ubuntu is recommended for beginners because of its popularity and it has a large resource of information on the web.

1 Introduction to Command Line Interface

The aim of this manual is to get you acquainted with the **Command Line Interface** of Linux systems. The Command Line Interface (CLI) is a powerful tool for performing various tasks on the system. Basically you enter a command from keyboard (which tells the computer to do something) and the computer executes your command, maybe interacting with you by displaying output or asking for additional input. When the command finishes executing, the system waits for another command from the user.

In Kubuntu, the Command Line Interface can be accessed by “Konsole” application. You may open this application by opening the application menu on the bottom-left corner of the screen and searching for “Konsole”. When you see the application, click on its icon.



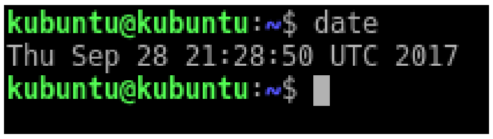
```
kubuntu@kubuntu:~$
```

Figure 2: Starting Konsole application in Kubuntu

The black screen you now see is the Command Line Interface. In it, you should see a single line of text. This is the “prompt line”; you may enter your commands when you see it. If command prompt does not appear after executing a command for any reason (program stuck in a loop, etc), you may hit **Ctrl-C** from keyboard to break the current program and forcefully return back to the prompt.

1.1 Basics

For example, lets try **date** command: type “`date`” from keyboard and hit Enter key:



```
kubuntu@kubuntu:~$ date
Thu Sep 28 21:28:50 UTC 2017
kubuntu@kubuntu:~$
```

Figure 3: Execution of **date** command

You should see that the computer has responded to your command by displaying the current day and time. Try date command a few seconds later again and you should see that the computer responds with a different output.

Try the following commands one by one:

- **whoami** - displays the username of the current user.
- **cal** - displays a calendar.
- **uptime** - displays how long the system has been running.
- **echo** - tells the computer to repeat a string:
  - **echo Hello World**
  - **echo $USER** - What do you see on screen? Why?
  - **echo Hello $USER** - Combination of previous two examples.
  - **echo Two plus five is $((2+5))** - The CLI may help you with mathematics too.

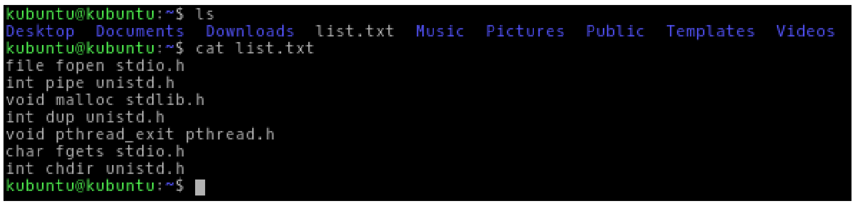
1.2 Working With Files

Echo command can also help you with creating text files. Enter the following commands:

```
echo file fopen stdio.h > list.txt
echo int pipe unistd.h >> list.txt
echo void malloc stdlib.h >> list.txt
echo int dup unistd.h >> list.txt
echo void pthread_exit pthread.h >> list.txt
echo char fgets stdio.h >> list.txt
echo int chdir unistd.h >> list.txt
```

Remember that when you used echo command previously, the output was written to the screen. But if you use ">" or ">>" signs, the output will instead be written to a specified file (list.txt in this case). Do note that using ">" will clear everything in the file and start from beginning, while ">>" appends to the end of file without clearing previous contents. This is analogous to the difference between opening files with "w" or "a" modes with **fopen()** function in C programming language.

Now that you have created a file, lets see if it is really there. Type "`ls`" command. This command lists available files and directories. Along with some directories, you should see your newly created file too. If you want to view contents of this file, you must execute "`cat list.txt`":



```
kubuntu@kubuntu:~$ ls
Desktop  Documents  Downloads  list.txt  Music  Pictures  Public  Templates  Videos
kubuntu@kubuntu:~$ cat list.txt
file fopen stdio.h
int pipe unistd.h
void malloc stdlib.h
int dup unistd.h
void pthread_exit pthread.h
char fgets stdio.h
int chdir unistd.h
kubuntu@kubuntu:~$
```

Figure 4: Execution of **ls** and **cat** commands

Execute the following commands one by one and understand how they work:

```
wc list.txt               - Count words, lines and characters in list.txt
grep stdio list.txt       - Search for the word ‘‘stdio’’ in ‘‘list.txt’’ and
                            display the lines in which this word appears
sort list.txt             - alphanumerically sorts lines in list.txt file and
                            display results on screen
sort list.txt > sorted.txt - sorts list.txt into sorted.txt file
cp list.txt os.txt        - make a new copy of list.txt named as os.txt
mv os.txt fun.txt         - rename os.txt into fun.txt
```

1.3 Working With Directories

In Linux based operating systems, files and directories are stored in an hierarchy as demonstrated below:

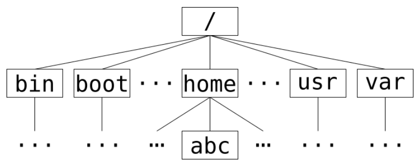


Figure 5: File System Hierarchy

When you first open Konsole, you usually start in your **home** directory, which resides in "`/home/`" directory (e.g. `/home/abc/`, `/home/student/`, etc). All files and directories owned by your account are stored in your home directory by default.

Execute the following commands one by one and understand how they work:

```
pwd            - to view in which directory you are working now
ls             - to view which files and directories are available
                 in your current directory
cd Desktop     - change current directory to Desktop
pwd            - to view in which directory you are working now
mkdir lab1     - make a new directory named lab1
ls             - to view which files and directories are available
                 in your current directory
cd lab1        - change current directory to lab1
pwd            - to view in which directory you are working now
ls             - to view which files and directories are available
                 in your current directory
cd ..          - go back to parent directory
pwd            - to view in which directory you are working now
cd ..          - go back to parent directory
pwd            - to view in which directory you are working now
```

You can move your files and directories into other directories. For example, execute the following:

- `mv fun.txt Desktop/lab1/`
- `mv sorted.txt Desktop/lab1/`

These two commands will move both your text files into lab1 directory you have created earlier.

1.4 Removing Files and Directories

Go into your lab1 directory by properly using pwd and cd commands. Then execute the following commands:

- **rm fun.txt** - removes fun.txt file.
- **rm sorted.txt** - removes sorted.txt file.
- **cd ..** - go to parent directory.
- **rmdir lab1** - remove lab1 directory.

Do note that rmdir requires the target directory to be completely empty before removing. Be careful! Using rm and rmdir commands will IMMEDIATELY delete files and directories, so you can not recover them back.

1.5 Getting Help

You may get a detailed help document for any command with **man** utility. For example try "`man ls`" now. You can scroll up and down using arrow keys. After reading the document, you may go back to command prompt by hitting **q** key.

You may get detailed documentation about C programming functions with man utility as well. Try "`man 3 printf`", "`man 3 scanf`", "`man 3 strcpy`", "`man 3 islower`", etc.

1.6 Permissions

Executing "`ls -l`" will generate a detailed list of information about your files and directories:

```
permission   owner   group   size  date         filename
drwxr-xr-x 5 student student 20480 May 21 13:47 Desktop
-rw-r----- 1 student student 141   Feb 12 10:50 lis.txt
```

The permissions are is divided into 4 parts:

- The first position specifies what type of file this is. - means regular file, d means directory, etc.
- The next three parts are divided into three sets of people: **u**ser, **g**roup, **o**thers. Each group gets three kinds of permission: **r**ead, **w**rite, e**x**ecute. For example lets focus on list.txt:

  ```
  user group others
  421  421   421
  rw-  r--   ---
  6    4     0
  ```

- "Access Mode" of this file is 640. It is calculated as "4+2+0, 4+0+0, 0+0+0".
- The first three bits specify the permissions for the user who owns the file. In this example, the owning user can read and modify this file.
- The next trio of bits are related to other people in the same group as the owner. In this example, group members may read this file but not modify it.

- The last trio of bits are for people outside the owner's group. In this example, they can neither read, nor modify this file.

You can alter access modes of files at any time. Execute the following commands and use "`ls -l`" to observe what has changed after each chmod command:

```
chmod 400 list.txt     - set read for owner and nothing else.
chmod 644 list.txt     - set read and write for owner, read for group and others.
chmod 000 list.txt     - remove all access to this file.
chmod +r list.txt      - give read access for all users.
chmod u+w list.txt     - give write access only to owner.
chmod o-r list.txt     - remove read access from others.
```

1.7 Command Line Scripts

Instead of entering commands one by one, you may store them in **script** files and execute them at once at any time later. Open Kate application from Kubuntu application menu and type the following:

```
#!/bin/bash

# this is a simple script for greeting the user
# and displaying working directory and date

echo "Hello $USER! You are now in:"
pwd
echo "The date and time is:"
date
```

Save this file as "hello.sh" in your Desktop. Now open a new Konsole and execute the following:

```
cd Desktop           - go into Desktop directory
chmod +x hello.sh    - give execution permission to hello.sh file
./hello.sh           - execute your script
```

You may define variables in your scripts (fav.sh):

```
#!/bin/bash

COURSE="CENG 328"
echo "My favorite course is $COURSE"
```

You may also perform more complex tasks with loops, if statements, etc. For example, the following script lists all files in the current directory and lists files which have their names starting with "ceng" but not "ceng2" (cenglist.sh):

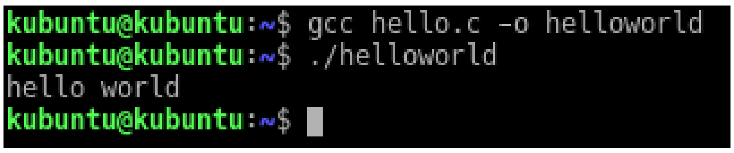
```
#!/bin/bash

for i in *; do
    if [[ $i == ceng* && $i != ceng2* ]]; then
        echo $i;
    fi
done
```

1.8 Compiling C / C++ Programs

You may develop programs using C programming language (and MANY more!) in Linux too. As a demonstration, open Kate from application menu and write a simple hello world program in C. Save your file as "hello.c" and type the following in a new Konsole window:

- `gcc hello.c -o helloworld` - Compile hello.c source code into helloworld program file
- `./helloworld` - execute the program



```
kubuntu@kubuntu:~$ gcc hello.c -o helloworld
kubuntu@kubuntu:~$ ./helloworld
hello world
kubuntu@kubuntu:~$
```

Figure 6: Compiling and Executing a C Program

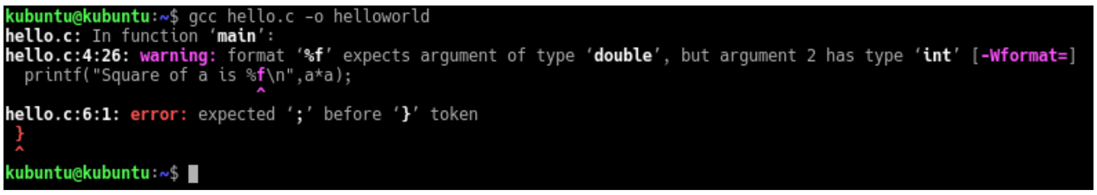
The gcc command translates your C source file into executable machine code. If you list files after executing gcc commmand, you should see two different files:

```
-rw-rw-r--  1 student student     65 Sep 29 12:56 hello.c
-rwxrwxr-x  1 student student   8600 Sep 29 12:58 helloworld
```

Notice that the new file, helloworld, has execution permissions.

Do note that in order to compile C++ programs, your source file must have either "**cc**" or "**cpp**" extension and you must use "**g++**" instead of "**gcc**".

Sometimes you may make mistakes in your programs. In such cases, gcc may display either a warning or an error message:



```
kubuntu@kubuntu:~$ gcc hello.c -o helloworld
hello.c: In function ‘main’:
hello.c:4:26: warning: format ‘%f’ expects argument of type ‘double’, but argument 2 has type ‘int’ [-Wformat=]
  printf("Square of a is %f\n",a*a);
                         ^
hello.c:6:1: error: expected ‘;’ before ‘}’ token
 }
 ^
kubuntu@kubuntu:~$ 
```

Figure 7: Compilation Warnings and Errors

If you encounter a warning message, that means your code **has compiled**, but may not work properly (e.g. runtime errors). If you encounter an error message, that means your code **has not compiled**. Fixing warnings is **recommended**, but fixing errors is **compulsory**.

1.8.1 Compiling Programs From Multiple Source Files

Sometimes you may need to compile programs that has its source code spanning across multiple source files. For example Figure 8 shows a portion of Mozilla Firefox source:

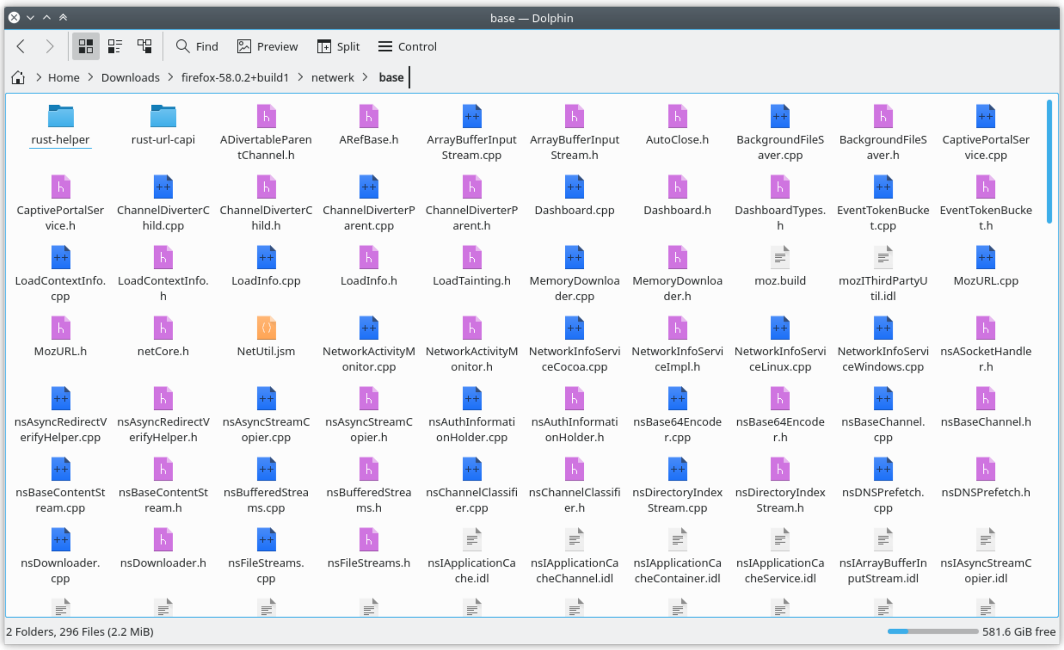


Figure 8: Multiple Source Codes

In order to compile more than one source files into one executable, you simply append every source file to your gcc line:

```
gcc source1.c source2.c ... -o program
```