Exercise 0: Basic Tools

This exercise is created so everybody can learn the basic tools we will use during this course. It is really more like a tutorial than an exercise and, you are not required to submit your results. The student assistants are in the lab to help you, not to check on your progress. The exercise is optional, but it is recommended that all complete it, so make sure you are prepared for the rest of the course.

1. The GNU/Linux environment

The computers on the real-time lab have two different operation systems; Windows and Ubuntu. When the computer boots, a menu where you can select between these is presented and you should select "Ubuntu".

1.1. Ubuntu

When Ubuntu is started you will get a login window. You should use "student" as the user name and "Sanntid15" as the password. If you don't already know Ubuntu, you should play around with the user interface. You should be able to find the programs for the following tasks:

- Web browsing (Firefox)
- Office programs (Open Office)
- Editors (Gedit, Emacs, Vim, Nano, Kate)
- Command line (text) interface (Terminal)
- File browsing

1.2. Shell

Even if Ubuntu has a graphical user interface, there are still many operations that are easier, faster or necessary to do from a command line interface (CLI). You can get a window with a CLI by starting the program *Terminal*.

The CLI is provided by one of several shell programs, most widely used are *bash*, *csh* and *tcsh*. The GNU/Linux default is *bash*, and you can check what type of shell your environment uses with the following command:

```
$ echo $SHELL
```

The first $-sign above is there just to indicate that the following text is a command to be type in the CLI. It should therefore not be written in the shell. Also remember to hit ENTER (this you know of course).

There are many other environment variables such as $PATH (where the shell searches for commands) and $HOME (the user home directory). In bash you can use the *env* and *export* commands to view and modify the environment variables.

In the shell find out more about *apropos, help, info,* and *man* by typing for instance:

```
$ man man
```

Hitting 'q' in *man* will exit the program.

Find out what the following commands do: *ls*, *cd*, *pwd*, *cp*, mv and *rm*

Find a command that tells you the date and the time on the system.

1.3. Directories and files

Your home directory is `/home/student`. Since this directory is shared with other groups using the same computer you should create the following structure: `/home/student/gruppeXX/ex1/` using the command *mkdir*.

You can create a simple text-file by using *echo* and the shell:

```
$ echo -n "I do really love" > test.txt
$ echo " TTK4147" >> test.txt
```

Try *du*, *file*, *cat* and *wc* on the file you just created.

```
$ du -h test.txt
$ file test.txt
$ cat test.txt
```

1.4. Mounting a network directory

To mount your NTNU home directory to a local folder on disk, enter the file browser and use the hotkey Crtl+L or Go->Location. In the Location: field enter the following:

```
//sambaad.stud.ntnu.no/<username>
```

Replace '<username>' with your NTNU user-name, hit enter, and fill in the username, domain(WIN-NTNU-NO) and password fields.

To unmount the network directory press the eject button on the location, beneath “places” on the left hand side of the window.

> *It is important that you back-up you data to a network folder (or some other media) prior to leaving the lab, and delete any files that you have stored locally. The computers may be reinstalled and all files deleted without any notice.*

1.5. Shell-scripts

It is possible to create small programs, or scripts, that are executed by the shell. Create a file named `script.sh` containing the following:

```
#! /bin/bash
for i in $(seq 1 10); do
   echo "Hello world $i"
done
```

Execute it by doing:

```
$ bash script.sh
```

or:

```
$ chmod +x script.sh
$ ./script.sh
```

What do the command *chmod* and the comment on the first line of the script do?

1.6. Jobs

To see the programs running on the system you can use *top, jobs and ps*. Do this now.

Execute the command *sleep 5*. Then try adding an '&' after the command. What happens? Try running *jobs* to examine running processes.

Start *sleep 5* again, and press CTRL+C while the process is running. Try again but use CTRL-Z instead. What is the difference between CTRL+C and CTRL+Z. What does the command *fg* do?

1.7. Editor

A programmer needs an editor to edit his code files. There are religious wars among programmers (and especially GNU/Linux programmers) about what is the best editor. We are not taking sides in this conflict, and you are free to use whatever editor you like. If you have no preference, and just want something that "just works", then *gedit* will probably be your best choice, while *nano* is an easy to use command line editor. Others editors are: *emacs, vim, kate* and *ed*.

2. Programming

2.1. GCC

The first Linux C-program you should create in this course is "Hello world". Use your editor to create the "Hello world" C-program shown below. Store the file `hello.c` in the working directory created for this exercise.

```
#include <stdio.h>

int main(int argc, char *argv[])
{
      printf(“Hello world\n”);
      return 0;
}
```

We will learn more about the GNU compiler tools in later exercises. For now, try to compile the program you just created with *gcc*:

```
$ gcc -o hello hello.c
```

Run the program with:

```
$ ./hello
```

2.2. Makefile

You should now create a more advanced program, inspect the following C code and figure out what it does:

```
#include <stdio.h>

int main(int argc, char *argv[])
{
      while(argc--)
            printf ("%s\n",*argv++);
      return 0;
}
```

Create a file name sort.c containing the following:

```
#include <stdlib.h>

void sort(int count, char *array[])
{
      int x,y;
      char* holder;
      for(x = 1; x < count; x++)
```

```
          for(y = 1; y < count-1; y++)
               if( atoi(array[y]) > atoi(array[y+1]) ) {
                    holder = array[y+1];
                    array[y+1] = array[y];
                    array[y] = holder;
               }
}
```

Make a call to the sort() function from the main() program, in order to sort the arguments list, and then print out the results.

```
#include <stdio.h>
#include "sort.h"

int main(int argc, char *argv[])
{
     int i;

     printf("Arguments before sort: ");

     for (i=1;i < argc;i++)
          printf("%s ",argv[i]);
     printf("\n");

     sort(argc,argv);

     printf("Arguments after sort:  ");

     for (i=1;i < argc;i++)
          printf("%s ",argv[i]);
     printf("\n");

     return 0;
}
```

The *sort.h* file should only contain a declaration of the function:

```
     void sort(int count, char *array[]);
```

You can compile the files by using:

```
$ gcc -o sort_args sort.c main.c
```

And run it:

```
$ ./sort_args 34 2 919 34 42 51353
```

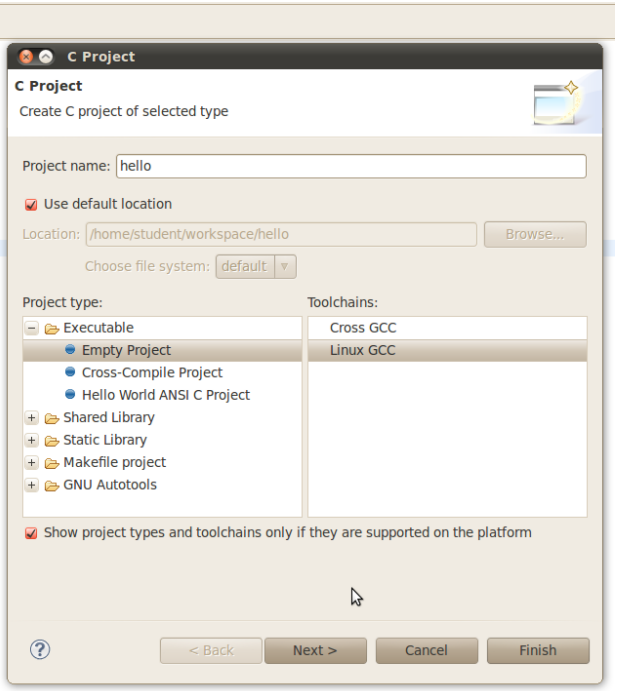
This is fine for smaller programs. But with larger programs and multiple files there is a problem: each time one of the files changes, all files are compiled again. A waste of computing resources! Instead, create the *Makefile* below. Make sure that there are TABs, not spaces in front of `gcc` and `rm`.

```
# program executable name
TARGET = sort_args

# compiler flags
CFLAGS = -g -Wall

# linker flags
LDFLAGS = -g

# list of sources
SOURCES = $(shell find -name "*.c")

# default rule, to compile everything
all: $(TARGET)

# define object files
OBJECTS = $(SOURCES:.c=.o)

# link programs
$(TARGET): $(OBJECTS)
	gcc $(LDFLAGS) -o $@ $^

# compile
%.o : %.c
	gcc $(CFLAGS) -c -o $@ $<

# cleaning
clean:
	rm -f $(TARGET) $(OBJECTS)
```

This Makefile can be used (with minor changes) for all Linux C programs we will develop in this course. It will compile and link all the source files in its folder. It will generate an executable that is names `sort_args`. To change this, edit the first line in the Makefile.

Build your program with the command `make`. Verify that the program is built correctly. Next, make some changes in either sort.c or main.c, and verify that only the changed files are re-compiled. You can also remove all generated files with `make clean`

3. Appendix: How to use Eclipse for programming and debugging C



In several of the exercises you will use the C programming language in the Linux operating system. You can use the tools you prefer. Some might prefer using their favorite text editor and makefiles for compiling the code, and this will be the default option described in the exercises.

Some might prefer to use an integrated development environment (IDE), like Eclipse. Eclipse was originally intended for Java development, but has extensions for use with C. The benefit of using Eclipse is that you have a code editor, build system and debugger all integrated in one program. The downside is that there are some additional steps for configuring it and that incorrect configuration can give unpredictable problems.

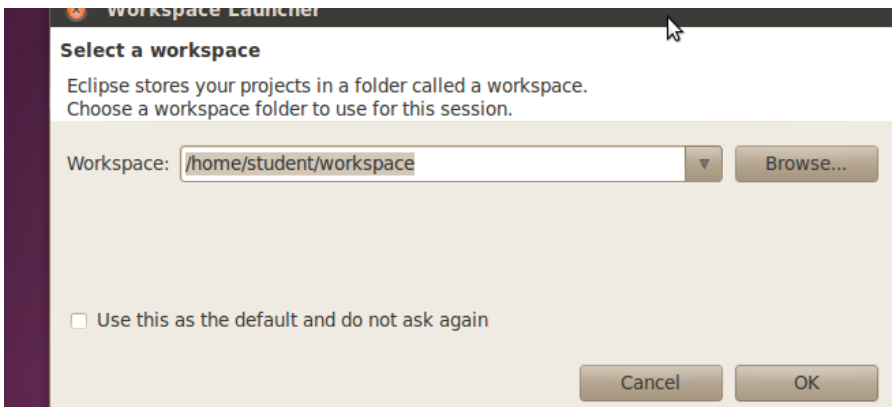
The basic set-up of Eclipse is described here. Necessary configuration changes for other exercises are described in appendixes at the end of the exercise text.

You start Eclipse by selecting “Eclipse C/C++” under programming in the application menu. You will be asked to specify a folder to be your workspace, as seen below. You should create a folder for this within your group folder.

To create a new project, click *File->New->C Project* to get the following window. Specify project name and select empty project and Linux GCC toolchain. Click finish.

Your new project needs a source file, click *File->New->Source File* and specify a name. You can also create header files like this. When you have created a small C program, you need to build the project, click *Project->Build Project*. If there are no errors, a folder called *Binaries* should appear in your Project explorer. You can now run the project by clicking *Run->Run As->Local C/C++ Application*. Eclipse will compile all the source files in your project, you don’t have to specify what to compile.

You can also use eclipse for debugging, you then click Run->Debug *As->Local C/C++ Application*. You will be asked if you want to open the debug perspective, which is probably a good thing, you can switch between perspectives in the upper right corner. Debugging in Eclipse allow you to easily define breakpoints, step through the code, look at variables etc.



4. Appendix: Subversion (svn)

Subversion (or other revision control systems) is very useful for development projects, and you can use it for the exercises in this course if you want to. It will make it easier to move between computers and also to check back on what you did earlier.

To get subversion on a NTNU server, you need to apply for a group account on the following website: http://www.stud.ntnu.no/kundesenter/. It could be that it takes a while to get the account, so you might not get access the same day you registered.

Now login to your NTNU account and do the following:

```
lab$ ssh user@login.stud.ntnu.no
ntnu$ cd /path/to/group/dir
ntnu$ svnadmin create svn
ntnu$ chown <username>:<groupname> svn –R
ntnu$ chmod g+rwX svn -R
ntnu$ exit
lab$ export
SVN=svn+ssh://user@login.stud.ntnu.no/path/to/group/dir/svn
lab$ svn mkdir $SVN/trunk -m "Creating trunk folder"
lab$ svn mkdir $SVN/trunk/01 -m "Creating folder for exercise 01"
lab$ svn mkdir $SVN/trunk/02 -m "Creating folder for exercise 02"
lab$ svn checkout $SVN/trunk group-XX
```

To find the full path to your SVN repository on the NTNU server use the *pwd* command. After this you should have a directory named "group-XX" (where XX is replaced with your group number). This folder has the subdirectories "01" and "02" for the first and second exercise respectively. You may now add the file from the previous exercise by copying them into the folder and doing the following:

```
$ cd group-XX/01
$ svn add my-report.ods
$ svn add my-file.c
$ svn commit -m "Added files for exercise 1"
```

Find what the following svn-commands does using *svn help*: add, delete, move, diff, copy, commit. You don't have to note this down; it's only to get acquainted with the system. Learn more about SVN here.

Put SVN in your professional toolbox, it's a gem. You will never regret to use revision control in any programming project, even if you're the only participant. It is not limited to code, it can also be used for handling documents and other files.