The Department of High Performance Computer Architecture
Prof. Dr. Volker Lindenstruth

Andreas Scholl (andreasscholl@stud.uni-frankfurt.de)
Kyrill Fries (kyrill.fries@stud.uni-frankfurt.de)

High Performance Computing (2019/2020)

Exercise 0

Introduction: Setup

In this class you will create programs written in C/C++ as well as evaluate their performace and improve them. For this to be possible every student needs an **Ubuntu 18.04 LTS** (`https://www.ubuntu.com/` installation since this will be the operating system your code is tested on. You are free to use whatever system you prefer to develope but make sure the version you hand in runs on the above specified system.

Virtual Machine

If you don't have a native Ubuntu installed, the easiest way to get access is via a virtual machine. Here you are also free to use whatever virtualization solution you prefer, but if you are unsure, we recommend VirtualBox `https://www.virtualbox.org/wiki/Downloads` since it is licensed under GNU General Public License (GPL) and there for free to use.

After you have installed the Ubuntu make yourself familiar with the commandline.

Listing files and directories

When you first login, your current working directory is your home directory. Your home directory has the same name as your user-name, and it is where your personal files and subdirectories are saved.

To find out what is in your home directory, type:

```
ls
```

The **ls** command lists the contents of your current working directory.
To list all files in long format within your home directory including those whose names begin with a dot (hidden), type:

```
ls -la
```

Making Directories

You will now make a subdirectory in your home directory to hold the files you will be creating and using in this course. To make a subdirectory called **hpc2019** in your current working directory type

```
mkdir hpc2019
```

To see the directory you have just created, type

```
ls
```

Changing to a different directory

The command **cd *directory*** means change the current working directory to 'directory'. The current working directory may be thought of as the directory you are in, i.e. your current position in the file-system tree.
To change to the directory you have just made, type

```
cd hpc2019
```

Type **ls** to see the contents (which should be empty)

The directories . and ..

Still in the **hpc2019** directory, type

```
ls -a
```

You will notice two directories, called . and ..
In Unix, . means the current directory. Command

```
cd .
```

changes the directory to the current direcory (the **hpc2019** directory).
This may not seem very useful at first, but using . as the name of the current directory will save a lot of typing.

Therefore, the command

```
cd ..
```

will take you one directory up in the hierarchy (in this case back to your home directory).

Pathnames

Pathnames enable you to work out where you are in relation to the whole file-system. For example, to find out the absolute pathname of your home-directory, type cd to get back to your home-directory and then type

```
pwd
```

The full pathname will look something like this -

```
/home/user/documents
```

which simply means that **documents** is in the sub-directory **user**, which is the **home** sub-directory, which is the top level root directory called "/".

Understanding pathnames

First type **cd** to get back to your home-directory, then type

```
ls hpc2019
```

to list the conents of your hpc2019 directory.

```
ls backups
```

You will get a message like this -

```
backups: No such file or directory
```

You can only list directories and files within the current working directory. To list another directory, which may be somewhere else, type

```
ls hpc2019/exercise
```

Home directories can also be referred to by the tilde   character. It can be used to specify paths starting at your home directory. So typing

```
ls ~/hpc2019
```

will list the contents of your hpc2019 directory, no matter where you currently are in the file system.

copy, move, etc.

To copy a file simply type **cp *file1 file2***, which makes a copy of **file1** in the current working directory and calls it **file2**.

```
cp /vol/examples/exercise/ex00.pdf .
```

With **mv *file1 file2*** you move or rename **file1** into **file2**. Compared to copy you end up with one file instead of two.

```
mv /vol/examples/exercise/ex00.pdf ~/hpc2019
```

To remove a file or directory use the **rm** command for a file and **rmdir** for a directory.

```
cp ex00.pdf tempex00.pdf
rm tempex00.pdf
```

The **rmdir** command will only remove empty directories, so make sure the directory you want to delete is empty.
Typing **clear** will clear the terminal window of the previous commands so the output commands can be clearly understood

```
clear
```

The command cat can be used to display the contents of a file on the screen. Type:

```
cat example.txt
```

Now you can see the files content without the need of opening it.
When a files content is bigger than the screen you will have to scroll. We do also have a command to display **less** content onto the screen a page at a time.

```
less example.txt
```

Press the space-bar if you want to see another page, type **q** to quit.

Searching the contents of a file

You can search a file for a keyword. Open the file using the **less** command and then type

```
less example.txt
/example
```

This will highlight you the first **example** keyword, type **n** to search for the next to occurrence.
However there is also a command to directly search a file for a specific keyword, the **grep** command.

```
grep example example.txt
```

grep will print out each line containing the keyword. Be careful because **grep** is case-sensitive! However you can also set options within the **grep** command, just type

```
grep --help
```

to get an overview of all the possible commands

Introduction: Compilers and build process

Install the gcc (Gnu C Compiler) using the following command:

```
sudo apt install build-essential
```

GCC

Compilation refers to the process of converting a program from the textual source code, in a programming language such as C or C++, into machine code.

Consider the following example:

```cpp
#include <iostream>
using namespace std;

int main(){
  cout << "Hello, world!!!" << endl;
  return 0;
}
```

Listing 1: Hello World example in C++ (hello.cc)

To compile `hello.cc` source file with gcc, use the following command:

```
g++ -Wall -o hello hello.cc
```

In this case, we use the '-o' option to specify a output file name for the executable, `hello`. The compiler does not produce any warnings with the `-Wall` option, since the program is completely valid.

To run the program, type the path name of the executable like this:

```
./hello
```

The path `./` refers to the current directory, so `./hello` loads and runs the executable file `hello` located in the current directory.

make

Compiling your source code files can be tedious, specially when you want to include several source files and have to type the compiling command everytime you want to do it. The trivial way to compile the files and obtain an executable, is by running the command:

```
g++ -o prg main.cc fun_a.cc fun_b.cc ...
```

Makefiles are special format files that together with the make utility will help you to automaticaly build and manage your projects.

Here is an example of `Makefile` for `hello.cc` example:

```makefile
CC=g++
# compiler options
CFLAGS=-c -Wall
# linker options
LDFLAGS=

# list of source files
SOURCES=hello.cc
# name of executable
EXECUTABLE=hello_make

OBJECTS=$(SOURCES:.cc=.o)

all: $(SOURCES) $(EXECUTABLE)

# link target (link all object files)
$(EXECUTABLE): $(OBJECTS)
        $(CC) $(LDFLAGS) $(OBJECTS) -o $@

# compile target (compile *.cc files to *.o object files)
.cc.o:
        $(CC) $(CFLAGS) $< -o $@
```

The basic makefile is composed of:

```
target: dependencies
#[tab] command to build the target
```

To build your project using `Makefile`, run the following command:

```
make
make -f <Makefile>
```

`make` will search the current directory for a file named `Makefile` (and some other names). To specify another `Makefile` run:

```
% make -f Makefile-other
```

Task 1

Create directory for exercise in your home directory, and extract prepared source files.

a) Compile and run the *hello world* example.

b) Create `Makefile` and build the `hello_make` executable.

Introduction: LaTeX

The Latex software package is a document preparation system that you will, at the latest, come in contact with during the preparation of your bachelor or master thesis. To provide you with some experience, we require that **all your exercise solutions** are to be created using LaTeX.

Install LaTeX

As with the previous software we do not care which environment you use. If you do not yet have LaTeX installed but have setup your Ubuntu installation, installing LaTeX is as easy as opening the commandline and typing:

```
sudo apt install texlive texlive-latex-extra
```

Compile PDF

After the installation download the homework_template.tar.gz, extract the latex-source and compile to produce the pdf.

```
tar xzf homework_template.tar.gz
cd homework_template
pdflatex homework_template.tex
```