Name:_____

Programming tools

Programming is the core of Open Source Software fundamentals, so it should come as no surprise that Linux distributions include a wide array of software development tools.

- sudo apt-get install build-essential git gitk devscripts

Unfortunately this exercise was written on a computer that had plenty of develoment libraries already installed. So if you encounter some library missing that was not mentioned in the instructions, consider it a part of the exercise to install those on demand. Doing these exercises should require no prior programming knowledge, but a whole lot of deduction and learning.

C and Git, an example project

1. First configure your name and email address to git. These will mostly make logs more pleasant to read.

   - *git config --global user.name "Your Name"*
   - *git config --global user.email your.name@aalto.fi*

   Git will save these to a file in your home directory ~/.gitconfig. You can also edit the data there using a text editor.

2. Create a project! Make a directory called "myown" and move there: *mkdir myown ; cd myown*

3. Create a new git repository: *git init*

4. Create a new source code file: *nano helloworld.c* Write the following into the code file you created:

   ```
   /* Hello World program */
   #include <stdio.h>
   int main(void) {
           printf("Hello World\n");
           return 0;
   }
   ```

   Beware of typos. Programming languages are universally ruthless about syntax errors. C programming is not part of this exercise or this course. For that there is *ELEC-A7110 C-ohjelmoinnin peruskurssi*.

5. Try compile your program with command: *gcc -o helloworld -Wall helloworld.c* (-o tells the output file and -Wall asks compiler to report all warnings)

6. Run the program you compiled: *./helloworld* ("./" in the beginning of command name means that you mean the program in your current directory. Otherwise you would get a warning that the program with that name is not found on the path.)

7. Commit this code file into the repository: *git add helloworld.c && git commit -m "New code file"* (-m "comment" is a way to leave a short comment to the commit. If you leave it out, git will start a text editor and let you write a commit log entry. The part && means that the latter command is executed only if the first command executes successfully.)

8. You can see the change you made in the log with command: *git log*

9. Now, the output of your fine program makes punctuationfetisists angry. Use text editor to change your program so that instead of "Hello World" it prints "Hello, World!".

10. You can observe the change you made with commands *git status* and *git diff*.

11. Commit the change: *git commit -m "Punctuation fix" helloworld.c*

12. Let's add a Makefile to the project to make it simpler to compile. Use text editor to create file "Makefile" and add following contents:

```
helloworld: helloworld.c
        gcc -o helloworld -Wall helloworld.c
```

13. Compile your program using command: *make*

14. Commit the Makefile: *git add Makefile*
git commit -m "Added Makefile"

15. Now that we consider the issue, the demands of punctuationfetists should not limit our academic freedom. Undo the change the wanted! Let's use the version management power of git to do this. Check from *git log* command output what was the commit id of the commit "Punctuation fix" (the long hex string after word commit) and run command: *git revert hexstring* Git will open a text editor where you can make the commit message, but let's assume the default message is good enough and close the editor.

16. Compile the program with *make* command and verify that it works: *./helloworld*

17. See what your git repository looks like with command: *gitk*

Makefile is a flexible mechanism to compile even larger projects. It's not actually tied to programming languages, but can work on multitude of things. (For example, compile Latex documents.) The syntax of rules is:

```
target: sources
       commands to turn sources into the target
```

A more comprehensive tutorial about Makefiles is available at: `http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/` And the complete documentation: `https://www.gnu.org/software/make/manual/make.html`
Git is distributed version management system. It can easily track changes made to the code. Word "Distributed" here means that multiple persons can change the source files simultaneously. If the changes affect different files (or differents parts of same file), they will "merge" automatically. Complex situations will require human intervetion to clean up.
Git documentation: `http://git-scm.com/documentation`

Compiling someone else's code

Most often you have to use code written by someone else. So let's get familiar how this works:

1. Get source codes for xbill (`http://www.xbill.org/`).

2. Install the library xbill uses for graphical interface (libxaw7-dev xaw3dg-dev).

3. xbill uses mechanism called "autoconf" to configure the program to your computer. You can tell this by presence of script named "configure". Run this script: *./configure*

4. Try compiling the program: *make*

5. Unfortunately there was a bug in the code and it was missing an instruction to link in libXpm library. Fix this bug by opening "Makefile", finding the line starting with word LIBS and adding "-lXpm" tothe end of that line.

6. Now compile the program (*make*) and run it: *./xbill*

   (Omnez has a fixed version of the code which you can get: *git clone https://gitlab.com/omnez/xbill21.git* He's been sloppy and included backup files and object files to the repository. This is not good style..)

Same again

7. xbill is also packaged with Debian. You can get the source code with command: *apt-get source xbill*
8. Move to directory xbill-2.1 and run: *debuild*
9. If you look into the directory "debian/patches" you find the patches that Debian project uses when compiling the program, including the libXpm fix we did above.

The process will create a deb package to the directory above your working directory. You can install the package (*dpkg -i xbill_2.1-8_amd64.deb*), but you will receive a warning that you are missing a GPG signing key. GPG-keys are used by Debian packaging mechanism to cryptographically guarantee that packages are official. We naturally don't have access to those keys since we're not Debian admins. The whole system of packaging is a whole lot larger than we can cover here. If you're interested, the Debian Project has massive amount of documentation at: `https://wiki.debian.org/Packaging`

Git+CMake example

This is an example of fetching and compiling a larger project:
`https://github.com/raceintospace/raceintospace`

1. Clone the project: *git clone https://github.com/raceintospace/raceintospace.git*
2. Install the libraries that the program needs: *sudo apt-get install cmake libsdl-dev libboost-dev libpng-dev libjsoncpp-dev libogg-dev libvorbis-dev libtheora-dev libprotobuf-dev protobuf-compiler*
3. Let's make a directory where we will do the compiling (so-called out-of-tree build):
   mkdir raceintospace-build
   cd raceintospace-build
   cmake ../raceintospace
   make

   The program should compile without any problems. Compiler will print out multiple warnings but they won't actually prevent the program from working.
4. When the program has been compiled and you would like to try it (it's an old space program game) you can install it to your system with command: *sudo make install*
5. If you want to remove the program, it will be installed in the directories /usr/local/share/raceintospace, /usr/local/bin/raceintospace and /usr/share/applications/raceintospace.desktop:

   sudo rm -rf /usr/local/bin/raceintospace /usr/local/share/raceintospace

   sudo rm /usr/share/applications/raceintospace.desktop

Note: You can easily host your own git repositories on any computer you can ssh into, but for shared repositories it's often useful to have some public sharing point. Github is one common service, but they are a for-profit company and will want you to make your repositories public (thus effectively advertising github) or pay for private repositories. Aalto University has it's own gitlab at `https://version.aalto.fi`.