1. Task in *Scientific Computing*

Deadline: March 29, 2021, 11:59pm

Survival Training in Linux

1. Linux commands and wildcards

Reading at home: Introduction[1] in wildcards, see also Wiki[2]. [3 pts]
Some commands and programming structures of the Bash[3].
More info for a single command, e.g., `find` via `man find` .

(i) Generate one new directory containing two subdirectories with simple C++-codes of you in these subdirectories.

- List all files with suffix *.cpp* in one of the subdirectories (wildcards).
- Return to the root directory.
- Change the access rights of the new directory and its subdirectories such that exclusively its owner/user is allowed to read the contents.

Hint: `mkdir, cd, ls, touch, gedit, pushd, popd, chmod`

(ii)
- List all files in your directory tree. Store that output in a file (redirect output).
- List all files in your directory tree with suffix *.h*.
- Set for files in your directory tree with suffix *.cpp* the timestamp to the current time.

Hint: `find`, `>`, `find ... -exec ...`

(iii) What is listed by the command `ls */*/*.[ch]*` if you call it after subtask (i)?

(iv) Generate sorted list of all files in a directory by using `ls` with sorting according to

(a) file size,
(b) file name,
(c) modification time,
(d) reverse modification time (newest file at last).

(v) Untar the archive File[4], via `tar` .

- List all the data in the archive file and store that list in a file.
- Determine how much disc space is used by the new directory.
- Determine how much disc space is used by the single subdirectories.
- Delete all *.log*, *.o*, *main.GCC_* files as well as the *html* directories.

Hint: `tar`, `du`, `find ... -o ... -exec ...`, `du`

(vi) `cd Kurs_C/Script/Beispiele`

[1] `http://ryanstutorials.net/linuxtutorial/wildcards.php`
[2] `https://en.wikibooks.org/wiki/A_Quick_Introduction_to_Unix/Wildcards`
[3] `http://arachnoid.com/linux/shell_programming.html`
[4] `http://imsc.uni-graz.at/haasegu/Lectures/SciComp/SS20/kurs.tar.gz`

- Compile *Ex433.cpp* using `g++`
- Edit the file such that a compilation error will will appear and redirect that output to a file *out.txt*.
- Redirect the output such that it will appear in terminal as well as in the file.

Hint: `g++`, `&>`, `>&`, `2>&1 |`, `tee`, `> out.txt 2>&1`

2. Regular expressions

Reading at home: Tutorial[5], wiki[6], and interactive testing[7] (Cheatsheet!) of regular expression. [2 pts]

(i) `cd Kurs_C/Script/latex`

- Find all tex-files containing the string *Alternative*
- Find all tex-files containing the strings *lauf* or *Lauf*, print the appropriate filenames as well as row numbers where the strings have been found.

Hint: `grep`, `fgrep`

(ii) The same directory as above.

- Replace in *p_7.tex* all German words *Funktion* by *function*. Take into account capital and lowercase.
- Delete from *p_7.tex* all comment lines (row starts with `%`) and store the result in *t7.tex*.
- Compare both files using `wc`, `diff`, `mgdiff`.

Hint: `grep`, `sed s/regex_in/regex_out/g ...` (see sed[8])

(iii) The same directory and file as above. Delete all comments until the end of the row. (∗)

(iv) The same file as above. Find all graphic file *.eps* or *.eps.gz* used in the file. (∗)

3. Shell scripts

Reading at home: Introduction[9] and tutorial[10] for bash scripts. [6 pts]

(i) Combine your solutions from tasks 1. and 2. into small bash scripts and run them.
Define variables in the script.
Use programming structures of the bash.
Hint: `for`, `while`, `if`

(ii) Store all files of a directory tree in an output file.
Look for certain file names in that output file, e.g., all files with suffix `.cpp` or `.h` .
Hint: `ls`, `grep`

(iii) Find all files larger as 500kB that haven't been used (accessed) for the last 3 months.
Hint: `find`[11]

[5] `http://www.regular-expressions.info/`
[6] `https://en.wikipedia.org/wiki/Regular_expression`
[7] `http://regexr.com/`
[8] `http://www.grymoire.com/Unix/Sed.html`
[9] `http://ryanstutorials.net/linuxtutorial/scripting.php`
[10] `https://linuxconfig.org/bash-scripting-tutorial`
[11] `https://shapeshed.com/unix-find/`

(iv) Unpack the archive file[12].

- Compare source files in `bsp_16*/*` from *Codes* regarding equality and and print appropriate directories and files.
  Hint: `diff`, see also `meld` when using a GUI.
- Check files `bsp_16*/*` in *Codes* with respect to certain key word (`break`, `continue`, `goto`) and assign the credits/points automatically.
  Hint: `grep`, `||` , `&&` , `eval`
- Archive your directory automatically (update !?). Compress the archive afterwards.
  Hint: `tar`, `gzip`, `zip`

(v) Convert strings in multiple files. Hint: `sed`

(vi) Unpack the archive file[13].

- Rename files in directory *gif*, e.g., *021906.gif* (mmddyy.gif) into *2006_02_19.gif* (yyyy_mm_dd.gif)
  Hint: `cut`, `mv`
- Shrink all images in that directory and store them as png files.
  Hint: `convert`

(vii) Write a shell script that calls your code (e.g., Goldbach) with different $n = \{10.000\ 100.000\ 400.000\ 1.000.000\ 2.000.000\ 10.000.000\}$ vi command line parameters and stores the timing for them. Visualize the graph ($time(n)$) automatically with appropriate tools (`gnuplot`, `octave`).

4. Makefiles

Reading at home: Tutorial[14] for `Makefile`s. [5 pts]

(i) Write a simple `makefile` for a C++-code of your own with at least two source files and one header file.
Your `makefile` has to realize compiling and linking (as separate steps): $.cpp \xrightarrow{\text{compile}} .o \xrightarrow{\text{link}}$ code.
See the Quick Reference[15].

(ii) Log the compiler output into file(s) such the compiler warnings etc. can be assigned to the appropriate source file.

(iii) Take into account that header files might have a newer time stamp than source/object files. All source files containing that header have to be recompiled.
Hint: `grep`, `touch`, `g++ -M`, `g++ -MM -MP`

(iv) Add new targets `makefile`:

- Delete all automatically generated files.
- Test the code.

[12] `http://imsc.uni-graz.at/haasegu/Lectures/SciComp/SS20/Codes.zip`
[13] `http://imsc.uni-graz.at/haasegu/Lectures/SciComp/SS20/gif.zip`
[14] `http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/`
[15] `http://www.gnu.org/software/make/manual/html_node/Quick-Reference.html#Quick-Reference`

- Pack all needed files into an archive such that a `make testing` after unpacking that archive performs compilation, linking, running and comparison to a reference output.
- Incorporate tools `valgrind` (memory checker) and `gprof` (profiler).
- Document your code with `doxygen` (`make doc`).
- Synchronize your code/archive with a server ..., via `git, scp, rsync, sitecopy` [whatever is available]

(v) Write a similar `makefile` for *Kurs-C/Script* that generates the pdf file, cleans and archives it (generate html?).

5. git

- Add the following lines to your file `~/.bashrc`

```
# git
# https://143.50.47.112:8443/
git config --global http.sslVerify false
git config --global user.email your_email@uni-graz.at
git config --global push.default simple
# git config --global core.editor "vi"
```

  and run `source ~/.bashrc` in the shell, or start simply a new shell.

- Check out the git-Repository [2 pts]
  `git clone https://<username>@143.50.47.112:8443/r/haase/lv/scicomp_21.git`
  and add Directory *your_familyname*. Copy one of your shell scripts into your directory, add/commit and push it into the global repository.

- Copy all your solutions from the tasks into your directory.
  Remove all unnecessary file **before** you add/commit/push them - especially the **binaries**/object files.

- Do not delete the directories of your class mates.

- Add the targets push/pop to your *makefile* of a project.

Literatur

[BaLi12] Daniel Barrett and Kathrin Lichtenberg. *Linux - kurz & gut*. O'Reilly, 2012. E-book[16].

[Ko07] Achim Köhler. *Der C/C++ Projektbegleiter*. dpunkt verlag, 2007. Link[17]

[La14] Hans Petter Langtangen. *A Primer on Scientific Programming with Python*. Springer, 2014. E-book[18].

[Wi04] Arnold Willemer. *Wie werde ich UNIX-Guru?*. Galileo Computing, 2004. OpenBook[19]

[16] `http://search.obvsg.at/primo_library/libweb/action/search.do?vid=UGR`
[17] `https://www.dpunkt.de/buecher/2584/9783898644709-der-c-c%2B%2B-projektbegleiter-10848.html`
[18] `http://search.obvsg.at/primo_library/libweb/action/search.do?vid=UGR`
[19] `http://openbook.rheinwerk-verlag.de/unix_guru/`

[Wo13] Christine Wolfinger. *Keine Angst vor Linux/Unix : Ein Lehrbuch für Linux- und Unix-Anwender*. Springer, Berlin Heidelberg, 2013. E-book[20].

[20] `http://search.obvsg.at/primo_library/libweb/action/search.do?vid=UGR`