Linux

BB1000 Programming in Python KTH

First essentials

Make a directory: `mkdir MY_FIRST_DIRECTORY` Create an empty file: `touch my_first_file` List all the files (and directories) in the current directory: `ls` Go into the directory MY_FIRST_DIRECTORY: `cd MY_FIRST_DIRECTORY` To show in which directory the cursor is located: `pwd` Go out of the directory: `cd ..`

```
$ pwd
/home/kronop
$ ls
$ mkdir MY_FIRST_DIRECTORY
$ ls
examples.desktop  MY_FIRST_DIRECTORY
$ cd MY_FIRST_DIRECTORY
$ touch my_first_file
$ ls
my_first_file
$ pwd
/home/kronop/MY_FIRST_DIRECTORY
$ cd ..
$ ls
examples.desktop  MY_FIRST_DIRECTORY
```

To remove an empty directory, `rmdir` is used. Files are deleted by `rm`.

Remark: for a python input cursor, we use `>>>` in this course - for a linux input cursor, we use `$`.

Move files

To rename files and to move files to another directory, the same keyword `mv` is used.

```
$ cd MY_FIRST_DIRECTORY
$ mv my_first_file my_first_other_name
$ ls
my_first_other_name
$ mv my_first_other_name ../
$ cd ..
$ pwd
/home/kronop
$ ls
examples.desktop  MY_FIRST_DIRECTORY my_first_other_name
$ mv my_first_other_name MY_FIRST_DIRECTORY/my_first_file
$ ls MY_FIRST_DIRECTORY
my_first_file
```

Copy files

Files are copied using `cp` - remark that the user always has to specify the destination…

```
$ cd MY_FIRST_DIRECTORY
$ cp my_first_file copy_of_my_first_file
$ ls
my_first_file copy_of_my_first_file
$ cp copy_of_my_first_file ../
$ cd ..
$ pwd
/home/kronop
$ ls
examples.desktop  MY_FIRST_DIRECTORY copy_of_my_first_file
$ ls MY_FIRST_DIRECTORY
my_first_file copy_of_my_first_file
```

Remove files

Keyword: `rm`…

```
$ pwd
/home/kronop
$ rm copy_of_my_first_file MY_FIRST_DIRECTORY/copy_of_my_first_file
$ ls
examples.desktop  MY_FIRST_DIRECTORY
$ ls MY_FIRST_DIRECTORY
$ my_first_file
```

"Hidden" files

Everything is governed by a text-file… Understand which file describes what!

With `ls -alh` all files in the directory are given - the normal files as well as the hidden ones which start with a `.`. The files are listed with their permissions, the name of their owner and the group of the owner.

```
$ ls -alh
drwxr-xr-x  3 kronop sknippen 4,0K nov 17  2016 .
drwxr-xr-x 31 root   root     4,0K feb  4 20:54 ..
-rw-------  1 kronop sknippen   59 nov 17  2016 .bash_history
-rw-r--r--  1 kronop sknippen  220 nov 17  2016 .bash_logout
-rw-r--r--  1 kronop sknippen 3,6K nov 17  2016 .bashrc
drwx------  2 kronop sknippen 4,0K nov 17  2016 .cache
-rw-r--r--  1 kronop sknippen 8,8K nov 17  2016 examples.desktop
drwxr-xr-x  2 kronop sknippen 4,0K feb 11 21:39 MY_FIRST_DIRECTORY
```

The response above can be compared with

```
$ ls
examples.desktop MY_FIRST_DIRECTORY
```

User kronop is a member of the group around user sknippen. The date of creation of the files is given (17 nov 2016), as well as the extend of the files (in Bytes).

Permissions

The permissions of the files (and directories if the first symbol is a `d`) are given in the first column. The first three symbols denote the usage for the owner of the file: `rwx` points at reading, writing and executing. The following three symbols denote the use for members of the group sknippen, of which the user kronop is a member.

In many cases, the group members are allowed to read the files (`r`) and eventually execute them (`x`). The last three digits denote the rights for users on the clusters which do not belong to the group sknippen. In any case, a normal user would not allow other people to change his/her files, so writing (`w`) is advised to be disabled for groupmembers and certainly for other people.

When a program is written, it is actually a text-file, which is made executable (`x`).

```
$ touch my_first_empty_program
$ ls -alh my_first_empty_program
-rw-r--r-- 1 kronop sknippen 0 feb 11 21:54 my_first_empty_program
```

Change permissions

The file `my_first_empty_program` is readable and writable for kronop, and it is just readable for the members of the group sknippen and for the rest of the users. It is not yet executable, therefore `chmod` is used.

`chmod u+x` renders the file for the user kronop to an executable file. `chmod g+x` gives also the members of the group the opportunity to run the file.

```
$ chmod u+x my_first_empty_program
$ ls -alh my_first_empty_program
-rwxr--r-- 1 kronop sknippen 0 feb 11 21:54 my_first_empty_program
$ chmod g+x my_first_empty_program
$ ls -alh my_first_empty_program
-rwxr-xr-- 1 kronop sknippen    0 feb 11 21:54 my_first_empty_program
```

Settings

In the .bashrc file, the personalized settings for the user kronop are defined.

From a cautious point of view, important settings could be:

```
alias rm="rm -i"
alias cp="cp -i"
alias mv="mv -i"
```

Thanks to these settings, linux will ask you whether you would really like to remove (`rm`) or overwrite (in case of `cp` and `mv`) files.

In case you write your own programs, you have to tell linux where they are located. Generally, a typical user like kronop has the habit to locate them in the /home/kronop/bin folder, which has to be loaded in the .bashrc file:

```
export PATH=/home/kronop/bin:$PATH
```

When the `.bashrc` file has been changed…

… you have to reload or `source` it so that the changes can take effect:

```
$ pwd
/home/kronop
$ source .bashrc
$
```

Text editor

`nano` is a basic text-editor. `nano filename` opens a file `filename`… At the bottom of the window, the various tools are given: '^X' points at 'ctrl-X' and means close (without saving). To save, 'ctrl-O' is used - afterwhich the texteditors asks a confirmation of the filename. If you don't want to save the information under another filename, press on enter. However, when you do want to change filename, `nano` will ask another confirmation (Yes/No).

Other commonly used text-editors are `gedit`, `vi` and `emacs`.

Finding files

It is a general advice to keep all your files ordered. It might be easy at this moment to find files back, but within three months, you should still know which file was connected with what project and how it was created…

However, to search files `find ./ -name` is used. The `./` points at the directory where (and down) the file is searched.

```
$ pwd
/home/kronop
$ find ./ -name my_first_file
./MY_FIRST_DIRECTORY/my_first_file
```

Searching information in a file

Imagine user kronop has to search the birth date of a friend Erik Ohlsson in a file which lists all the inhabitants of the village with their data.

```
$ cat birthdates_village
Olav Carlsson 1969-01-12
Emma Karlsson 1981-04-23
Erik Ohlsson 1972-12-3
Mohammed Puidgemon 1988-02-29
Helen Tantra  1945-09-15
(...)
```

`cat` prints the content of a file to screen - but it cannot be changed.

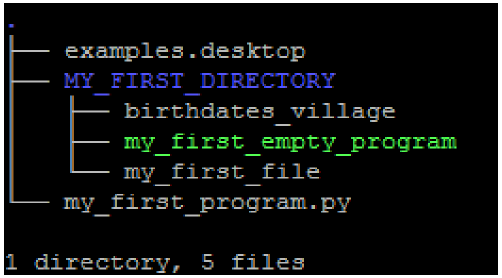
In stead of browsing through the file, the linux function `grep` can be used.

```
$ grep Erik birthdates_village
Erik Ohlsson 1972-12-3
```

What is the structure among my files?

At each instance in the hierarchy of the data, keyword `tree` can show the structure of the directories and the files at that position and below.

```
$ pwd
/home/kronop
$ tree
```



```
.
├── examples.desktop
├── MY_FIRST_DIRECTORY
│   ├── birthdates_village
│   ├── my_first_empty_program
│   └── my_first_file
└── my_first_program.py

1 directory, 5 files
```

When you don't know...

Linux has a built-in help function, which is especially useful when you do not know which optional parameters can be used... Simply write `man` followed by the command you would like to have more information about. `man ls` will give various pages of information about the `ls` command...

```
$ man ls

LS(1)                           User Commands                           LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the FILEs (the current directory by default).
       (...)
```