A beginners guide to LINUX

Keno Budde

January 10, 2017

Contents

1 introduction

Coming from a non-Unix background I must admit that I had some difficulties getting used to Linux. Nonetheless I can know see the advantages of using a UNIX system. Especially the possibilities of the command line amazed me, in combination with bash scripting. The only thing I think is a shame is that many programs are not available for Linux I really enjoy using daily. Otherwise I would have changed to a Linux distribution as my main operating system.

2 the command line

```
user@DESKTOP-8037VPV:/mnt/c/Users/Tammo/Documents/cs133$
```

Figure 1: now my favorite tool

The command line is the UNIX tool I felt the most uncomfortable with on my first day of using a linux distribution but it became the tool I enjoy now the most. What techrepublic can say about command line applications was exactly how I experienced my life with this tool: “Beginners can learn the rudimentary basics easily, and as they learn more about how to use it their facility with the application expands. ” [2] I began to enjoy the possibilities I would have using the terminal and the time I would gain in comparison to working on my windows operating system. I take pleasure in easily moving around files and directories with simple commands without having to drag and drop them around. The five commands I got the most comfortable with in the short amount of time I should call the first term are the following:

1. **ssh**: using ssh or the secure shell is something I got very used to especially when working. ssh allows a user to securely log into a remote system. Using the **-Y** option to enable X11 forwarding it is even possible to see a graphical remote desktop. An example for that command would be the way I could log onto the server of the department of computer science, Joshua.

   ```
   $ ssh -Y u1632823@joshua.dcs.warwick.ac.uk
   ```

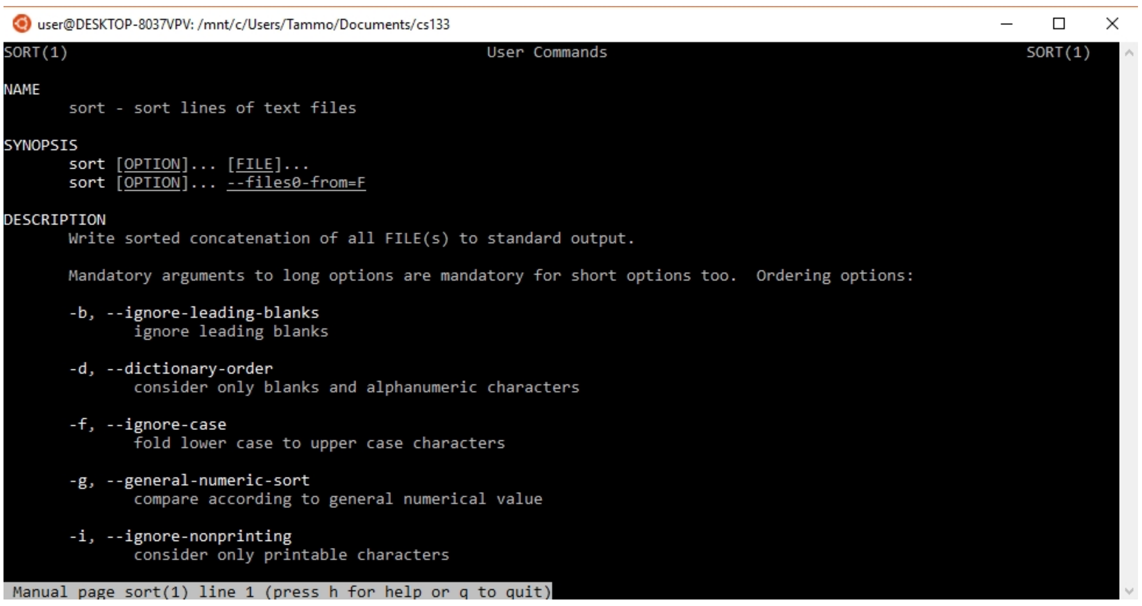
2. **git**: another very important program for me I want to talk about later in this guide is git. Using it as a version control tool saved my work multiple times when I could roll back to the last working version.

3. **man**: while trying to get to know Linux my best helper were manual pages of the commands I wanted to use. I first had to get used to the way they are written but now they are an essential asset for my work. Working on a bash program the other day I wanted to get to know the sort command, so the first thing I did was to consult the manual page of the command.

4. **lpr**: printing out my work for various assignments I really enjoyed the ease of use of the lpr command which enables the user to print a file easily. To actually print a document it is only necessary to specify the printer and the document I wish to print:

   ```
   $ lpr -Pdcs001a info.pdf
   ```

5. **cd**: an essential command to navigate is cd and its ease of use is wonderful. To navigate somewhere, it is possible to give the command an absolute path, which could be something like this:



```
user@DESKTOP-8037VPV: /mnt/c/Users/Tammo/Documents/cs133
SORT(1)                                          User Commands                                         SORT(1)

NAME
       sort - sort lines of text files

SYNOPSIS
       sort [OPTION]... [FILE]...
       sort [OPTION]... --files0-from=F

DESCRIPTION
       Write sorted concatenation of all FILE(s) to standard output.

       Mandatory arguments to long options are mandatory for short options too.  Ordering options:

       -b, --ignore-leading-blanks
              ignore leading blanks

       -d, --dictionary-order
              consider only blanks and alphanumeric characters

       -f, --ignore-case
              fold lower case to upper case characters

       -g, --general-numeric-sort
              compare according to general numerical value

       -i, --ignore-nonprinting
              consider only printable characters

 Manual page sort(1) line 1 (press h for help or q to quit)
```

```
$ cd /mnt/c/Users/Public
```

Another possibility to use this command is to give it a relative path to work with, an example is shown below:

```
$ cd ../Documents/cs132
```

3 version control

As I mentioned above over the last months I took a lot of pleasure from working with git as a version control software. It saved my work more than once and it was also very motivating for me to see the statistics of my progress online through my github.com calendar. After reading up a bit more about git and its possibilities as a tool for programmers I created an account on github to use it as a backup for the files I am working on. Working with git is really easy. The repository to work on has first to be intialized for git:

```
$ git init sample_directory
```

If if been working on this project before without starting git from the beginning I would have to add all files to git, otherwise I could specify specific files:

```
$ git add *
```

After that the first commit has to be made:

```
$ git commit -m "created directory and documentation files"
```

More often than not I am using a service like github.com to backup my work. This means that the remote service has to be set up:

```
$ git remote add origin url
# sets a new remote server
$ git remote -v
# verifies the new service
```

Another useful feature I learned about is the possibility to take a look at a previous commit. [1] Therefore the git log feature makes it easy to have a better look at the commits:

```
$ git log --oneline
```

By "checking out" to a previous commit it is possible to have a look at the way the code looked the time the commit was done.

```
$ git checkout commit
```

If I want to do an experiment I usually create a new branch to experiment on.

```
$ git branch experiment
# creates a new branch called experiment
$ git checkout experiment
# checks out the experiment branch which means that
# the user is working now on this branch
```

When I am happy with the experiments I made and tested them I can now reinclude those changes into the master branch. Therefore I can rebase now the experiment branch into the master branch and merge them together [3]:

```
$ git rebase master
# rebases the experiment branch to the master branch
$ git checkout master
# checks out the master branch
$ git merge experiment
# merges the two branches together
```

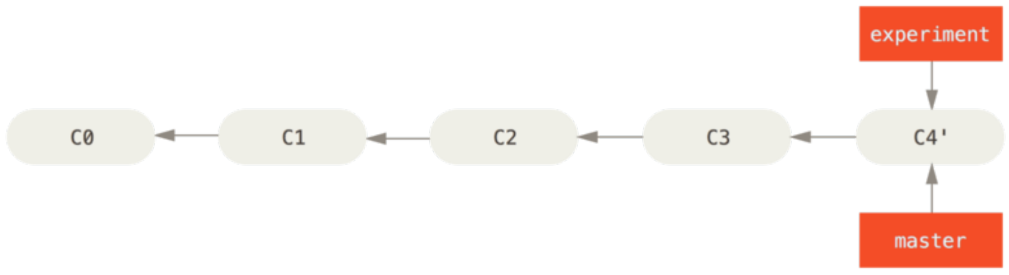


Figure 2: Forwarding the master branch [3]

4 useful commands and programs

Following up on the short list of the commands I mentioned in a previos section I want to talk about a few more useful commands and programs I enjoy using.

- **cp**: Enables the user to copy files.

```
$ cp index.html index2.html
# copies index.html into a second file
# called index2.html
```

- **mv**: Enables the user to move and rename files.

```
$ mv index.html index2.html
# renames index.html into index2.html
```

- **export**: Enables the user to set an environmental variable. For example this command could be used to set include a new manual page:

```
$ export MANPATH=$(man -w):~/lab1resources/man
```

- **mkdir**: creates a new directory. For example when I created the public_html directory I used this command:

```
$ mkdir public_html
```

- **chmod**: changes permissions of files or directories. Can be used with an octal number or the explicit access permissions:

```
$ chmod 644 index.html
# changes the permissions to read/write
# for the owner, read only for users
# in the group and others
```

- **ls**: lists files in the current directory. The command can be modified to list hidden file or to work recursively:

```
$ ls -all
# lists all files in the directory including
# hidden files
```

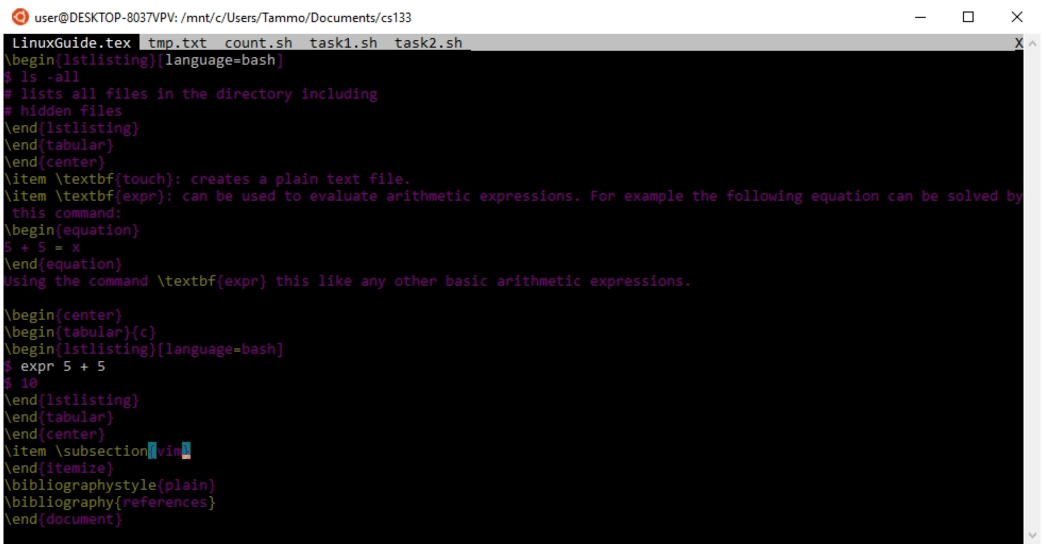
- **touch**: creates a plain text file.
- **expr**: can be used to evaluate arithmetic expressions. For example the following equation can be solved by this command:

$$5 + 5 = x \tag{1}$$

  Using the command **expr** this like any other basic arithmetic expressions.

```
$ expr 5 + 5
$ 10
```

4.1 vim



```
\begin{lstlisting}[language=bash]
$ ls -all
# lists all files in the directory including
# hidden files
\end{lstlisting}
\end{tabular}
\end{center}
\item \textbf{touch}: creates a plain text file.
\item \textbf{expr}: can be used to evaluate arithmetic expressions. For example the following equation can be solved by
 this command:
\begin{equation}
5 + 5 = x
\end{equation}
Using the command \textbf{expr} this like any other basic arithmetic expressions.

\begin{center}
\begin{tabular}{c}
\begin{lstlisting}[language=bash]
$ expr 5 + 5
$ 10
\end{lstlisting}
\end{tabular}
\end{center}
\item \subsection{vim}
\end{itemize}
\bibliographystyle{plain}
\bibliography{references}
\end{document}
```

Figure 3: My favorite text editor

The text editor I came to love over the last months is vim as a very easy to use command line tool. Vim can be started up with a specified file or on its on.

```
$ vim index.html
# starts up vim and opens the file index.html
$ vim
# just starts up vim on its on
```

At this point I want to talk about a few possibilities and modes vim provides:

- **Insert mode**: Starting up vim the user is in read mode which means the content of a file can not be edited. To edit the files content the insert mode has to be selected by typing **I**.
- **Selecting content in vim**: By escaping the inserting mode by pressing the **Esc** key a new mode can be selected. Opening up the visual mode by selecting **V** text can be selected.
- **Copying and Cutting text**: Selected text can be copied by using **y** and cut by using **d**. It can be inserted by using **p** to insert text after the line the cursor is pointing to and can be inserted before the current line using **P**
- **tabs**: On element of vim I enjoy a lot are tabs. A new document can be opened in a tab using **:tabe document**. There is the possibility to change between the tabs using **:tabn** to change to the next tab or **:tabp** to change to the previous tab.
- **using bash inside vim**: using **:!** it is possible to execute any bash command inside vim.
- **saving files**: Files can be written using **:w**.
- **quitting the program**: The program can be quit using **:q**.

References

[1] Atlassian. Viewing old commits. `https://www.atlassian.com/git/tutorials/viewing-old-commits/`. Accessed: 2016-12-30.

[2] Chad Perrin. Five benefits of command line tools. *techrepublic.com*, 2011. Accessed: 2016-12-30.

[3] Ben Straub Scott Chacon. *Pro Git*. Apress, second edition, 2014.