Linux & Bash

Michiel Noback

February 15 2017

The course contents

The topics of this course

- The file system
- Permissions and processes
- A selection of tools

The file system

- navigation
- structure
- permissions
- the bin folder
- hidden folders

The terminal

- executing commands
- man pages (arrgghhh)
- help
- piping
- redirecting

A selection of tools

- file operations
- file editing
- finding & filtering stuff (grep, find, sed)

Intro: Why use Linux / Terminal?

The power of CLI

- CLI = command line interface; typing text instead of using the mouse to give commands
- Modular
- Large number of programs avialable in your toolbox
- Easy to automate
- Easy to combine

Basic terminology

- The **terminal** is the wrapper program that runs the shell
- The **shell** is the program which actually processes your entered commands
- Shell is actually a specification; **bash** is an implementation of it (and there are others)
- Here, we will regard **console** as a synonym to terminal

structure of a CL command

- command (program name)
- options (usually in the form of flags like `-i`, `--verbose`)
- other arguments

Single versus double quotes

- double quotes are expanded by shell
- single quotes passed literally

```
echo "USER is $USER"
echo 'USER is $USER'
```

```
USER is michiel
USER is $USER
```

- use quotes when passing arguments with spaces

Part 1.1: The file system

The Linux file system (1)

- Linux organizes files in a tree-like pattern
- Each directory can contain other files and directories
- The first directory is called the *root* directory: / . .
- Linux has a single-filesystem tree, *regardless how many storage devices are attached* (mounted)
- The system contains many directories for "administration" purpose: /bin /boot /dev /lib /etc
- Each user has a home environment

The Linux file system (2)

Important root folders

- `/bin` – essential system executables (single user system)
- `/sbin` – superuser executables (single user system)
- `/usr/bin` – idem (multi user system)
- `/usr/sbin` – idem (multi user system)
- `/usr/local/bin` – local executables (usually simlinks)
- `/usr/local/sbin` – idem - for superuser

The working directory

In any terminal session, you will have a *working directory*. You get its current location by typing `pwd` (**p**rint **w**orking **d**irectory)

```
pwd
```

```
## /Users/michiel/OneDrive - Hanzehogeschool Groningen/cou
```

changing the working directory: `cd`

`cd` (**c**hange **d**irectory) can be used in several ways:

- `cd <absolute path>` *e.g.* `cd /homes/michiel/tmp/`
- `cd <relative path>`*e.g.* `cd ../tmp/foo`, `/data`
- `cd ..` to go up one level
- `cd -` to toggle between two directories

listing the contents: `ls`

- `ls` lists the contents of a folder
- if none is provided, defaults to current working directory.
- `ls` knows *many* options (type `man ls` to find out), but these are most relevant
  - `ls -a` also show hidden files (starting with a dot)
  - `ls -l` use long format (including date, size, rights)
  - `ls -R` list directory contents recursively
- use `ls -l <filename>` to get info on a single file

Combining flags

With `ls`, as with many CL tools, you can combine (short) flags into one string

This command

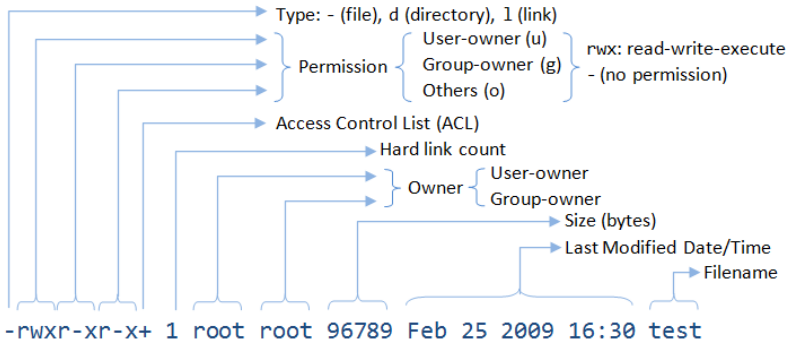
```
ls -l -a -h -G
```

is the same as

```
ls -lahG
```

The `ls -l` long format

The long format has this structure



```
-rwxr-xr-x+ 1 root root 96789 Feb 25 2009 16:30 test
```

`cd` and `ls` demo

Given this structure:

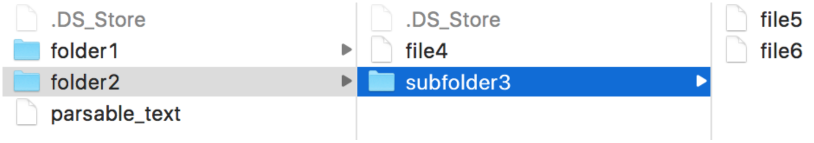


Figure 1: Folder Structure

Let's have a look at some navigation possibilities.

```
cd data/
ls
cd folder1
pwd
cd ../folder2 #same as two commands: cd .. and cd folder2
pwd
ls
```

```
## chsi_sample_measuresofhealth.csv
## data.zip
## error.log
## folder1
## folder2
## hello.log
## normal.log
## some_text
## streams_demo.py
## /Users/michiel/OneDrive - Hanzehogeschool Groningen/cou
## /Users/michiel/OneDrive - Hanzehogeschool Groningen/cou
## file4
```

Part 1.2: The file system: Working with Files

Inspection file contents

There are a few programs that support this, each with its own specifics

Use `less` to read and scan text

- `less` is used to browse and search text files
- it allows scrolling up and down using arrow keys or mouse
- it also supports keyword search: type `/keyword_to_search` and hit `ENTER` to do this
- `more` is its handicapped predecessor

Fetch a `head`-ing

- Use `head` to fetch the first few lines of one or more files
- Defaults to 10 lines
- Supports multiple files
- Fetching the first 3 lines from file `some_text` (contents copied from Wikipedia):

```
head -3 ./data/some_text
```

```
Linux is a Unix-like and mostly POSIX-compliant[12] compu
operating system (OS) assembled under the model of free a
open-source software development and distribution.
```

Fetch a file `tail`

- Same as `head`, but applied to the end of files

```
tail -4 ./data/some_text
```

```
facility automation controls, televisions,[26][27] video
consoles and smartwatches.[28]
...
source: Wikipedia
```

Note: you may have noticed this file has no extension; Linux does not really care about that. Use the command `file <filename>` to find out if unsure.

Use cat to echo the contents

- This is often used in pipe and redicrect operations, discussed later
- Can be applied to multiple files
- Often used to combine contents:

```
cat file1.txt > all.txt
cat file2.txt >> all.txt
```

Use `nano` to edit while in terminal

This is especially interesting when you have an SSH connection to another host. Read online instructions for this tool.

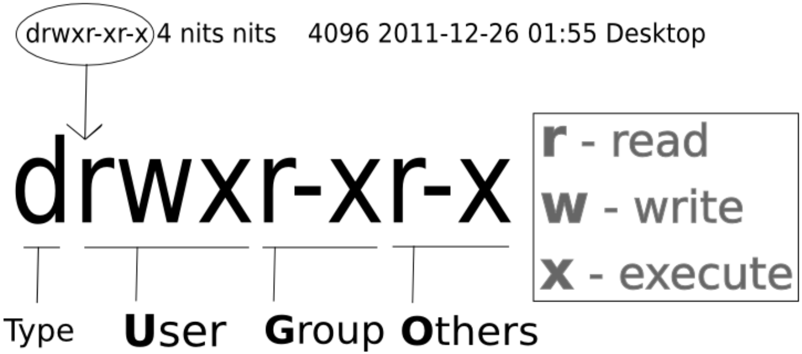
Part 2: Linux (file) system

Topics

- permissions
- environment variables, $PATH and bin folder

Permissions

- When viewing files with the `ls -l` options, you get the file *permissions*
- This is what it means



File permissions in Linux

Changing permissions

- To change the permissions, use `chmod`
- The following symbols are used to represent the users, groups and others:
  - `u` : User
  - `g` : Group
  - `o` : Others a : All (user, group and others)
- The following symbols represent the permissions grant or revoke:
  - + : Additional permissions. Selected permissions are added.
  - − : Revoke the permissions. Selected permissions are revoked.
  - = : Specific permissions. Only selected permissions are assigned.

Changing permissions

- The are some common ones:
  - `chmod a+r *.txt` grants read rights to everyone for txt files
  - `chmod a-x *.sh` revokes execute rights on all shell scripts
  - `chmod u+x app.py` makes file app.py executable for user

Environment variables

- Environment variables are variables like in any programming environment, but in this case for the terminal environment
- They are usually all-caps and are preceded by a dollar sign
- The most important one is $PATH, containing the locations to look for executable files (programs)

```
echo $PATH
```

```
/usr/bin:/bin:/usr/sbin:[....]/sbin:/usr/local/bin:/usr/tex
```

Changing the environment

- When you start a terminal, an **environment** is loaded from /etc/profile
- You can extend this behaviour using `.bashrc` (or `.bash_profile`)
- The next few slides outline some interesting extensions

.bashrc aliases

You can create aliases for your much-used commands (or tools you created yourself). Sometimes, you find them in a separate file called .bash_aliases. Here are a few examples:

```
alias duh='du -h -d 1'
alias ll='ls -FGlAhp'
alias ..='cd ../'
alias p='pwd'
```

.bashrc environment variables

Some programs require certain environment variables to be set.
Also, it may be nice to set some variables for yourself.

```
export JAVA_HOME=/usr/libexec/java_home
export PATH=".:/homes/michiel/my_cool_app/bin:$PATH"
export EDITOR=/usr/bin/nano
export PROJECT="/homes/michiel/../../my_current_project"
```

.bashrc configuring your prompt

The prompt is what you see on your current terminal line, waiyting for your commands. It can be very simple or very complex. Usually you will find (some of) these fields

- \u the current logged in user (you), same as $USER
- \h the host (machine) you are working on
- \w the current working directory

```
##with color
PS1="\[\033[36m\]\u\[\033[m\]@\[\033[32m\]\h:\[\033[33;1m\]
#simple, with a newline appended
PS1="\u@\h:\w\n\$ "
```

.bashrc configuring your prompt

Now play around with it, have a look at this site.
Store your original prompt like this `OLD=$PS1` and restore it like this `PS1=$OLD`

Part 3: Processes

What is a process

A process is an instance of a computer program that is being executed.

It contains the program code and its current activity.

What can you do with processes?

- starting (with delay)
- viewing
- killing
- redirecting output (see next presentation)
- piping (see next presentation)

Starting in the background

- Use the & symbol at the end of a command to start it in the background
- Use at to schedule it to run at a later time

```
geany my_script.py & ##starts geany with script in editor
```

Schedule a job with at

- Use at to schedule a job
- Start at with a time span, or date/time
- The given job will be executed and output mailed to your system email adress.
- See http://www.computerhope.com/unix/uat.htm for additional info
- Here are a few examples for specifying time:

```
at 9:30 PM Tue
now + 30 minutes
now + 1 hour
```

Schedule a job with `at` (2)

- Use `atq` to inspect your scheduled jobs, including their job numbers
- `atrm` deletes jobs, identified by their job number (use atq to see what the job number is!)

```
atrm 23 #removes at job 23
```

at examples (1)

- The first one schedules a job for 14:15, executing “chkdsk /f”
- The second one schedules a job for 1 hour from now, to execute the commands specified in file `shell_script`
- The third one will schedule a job for one minute from now, executing commands entered at the interactive `at` prompt

```
at 14:15 "chkdsk /f"
at -f shell_script now + 1 hour
at now + 1 minute
```

at examples (2)

Direct entry is done like this (finish entering commands by typing `ctrl + d`):

```
michiel@bin006$ at now + 1 minute
warning: commands will be executed using /bin/sh
at> echo "hello" > hello.txt
at> <EOT>
job 21 at Mon Feb 27 11:13:00 2017
##wait a minute
michiel@bin006$ cat hello.txt
hello
```

at examples (3)

Last example; output from stdout and stderr is sent to email address registered in `/.forward`:

```
michiel@bin006$ at now + 1 minute
warning: commands will be executed using /bin/sh
at> echo "Hello world!"
at> <EOT>
job 22 at Mon Feb 27 11:16:00 2017
```

Viewing and killing processes

- Use `htop` (or `top`) or `ps aux` to view and `kill` processes if required.