LINUX

BASH SCRIPTING

Internettechnologie

Academiejaar 2019-2020
Les 1
M. DIMA

katholieke hogeschool
associatie KU Leuven
vives

man pages

Most Unix files and commands have pretty good man pages to explain their use. Man pages also come in handy when you are using multiple flavours of Unix or several Linux distributions since options and parameters sometimes vary.

man $command

Type **man** followed by a command (for which you want help) and start reading. Press **q** to quit the manpage. Some man pages contain examples (near the end).

```
paul@laika:~$ man whois
Reformatting whois(1), please wait...
```

man $configfile

Most **configuration files** have their own manual.

```
paul@laika:~$ man syslog.conf
Reformatting syslog.conf(5), please wait...
```

man $daemon

This is also true for most **daemons** (background programs) on your system..

```
paul@laika:~$ man syslogd
Reformatting syslogd(8), please wait...
```

man -k (apropos)

man -k (or **apropos**) shows a list of man pages containing a string.

```
paul@laika:~$ man -k syslog
lm-syslog-setup (8)  - configure laptop mode to switch syslog.conf ...
logger (1)           - a shell command interface to the syslog(3) ...
syslog-facility (8)  - Setup and remove LOCALx facility for sysklogd
syslog.conf (5)      - syslogd(8) configuration file
syslogd (8)          - Linux system logging utilities.
syslogd-listfiles (8) - list system logfiles
```

whatis

To see just the description of a manual page, use **whatis** followed by a string.

```
paul@u810:~$ whatis route
route (8)             - show / manipulate the IP routing table
```

whereis

The location of a manpage can be revealed with **whereis**.

```
paul@laika:~$ whereis -m whois
whois: /usr/share/man/man1/whois.1.gz
```

man man

If you want to know more about **man**, then Read The Fantastic Manual (RTFM). *Unfortunately, manual pages do not have the answer to everything…*

```
paul@laika:~$ man woman
No manual entry for woman
```

working with directories

This module is a brief overview of the most common commands to work with directories: **pwd**, **cd**, **ls**, **mkdir** and **rmdir**. These commands are available on any Linux (or Unix) system.

This module also discusses **absolute** and **relative paths** and **path completion** in the **bash** shell.

8.1. pwd

The **you are here** sign can be displayed with the **pwd** command (Print Working Directory). Go ahead, try it: Open a command line interface (also called a terminal, console or xterm) and type **pwd**. The tool displays your **current directory**.

```
paul@debian8:~$ pwd
/home/paul
```

8.2. cd

You can change your current directory with the **cd** command (Change Directory).

```
paul@debian8$ cd /etc
paul@debian8$ pwd
/etc
paul@debian8$ cd /bin
paul@debian8$ pwd
/bin
paul@debian8$ cd /home/paul/
paul@debian8$ pwd
/home/paul
```

katholieke hogeschool associatie KU Leuven vives

8.2.1. cd ~

The **cd** is also a shortcut to get back into your home directory. Just typing **cd** without a target directory, will put you in your home directory. Typing **cd ~** has the same effect.

```
paul@debian8$ cd /etc
paul@debian8$ pwd
/etc
paul@debian8$ cd
paul@debian8$ pwd
/home/paul
paul@debian8$ cd ~
paul@debian8$ pwd
/home/paul
```

8.2.2. cd ..

To go to the **parent directory** (the one just above your current directory in the directory tree), type **cd ..** .

```
paul@debian8$ pwd
/usr/share/games
paul@debian8$ cd ..
paul@debian8$ pwd
/usr/share
```

*To stay in the current directory, type **cd .** ;-)* We will see useful use of the **.** character representing the current directory later.

8.3. absolute and relative paths

You should be aware of **absolute and relative paths** in the file tree. When you type a path starting with a **slash** (/), then the **root** of the file tree is assumed. If you don't start your path with a slash, then the current directory is the assumed starting point.

The screenshot below first shows the current directory **/home/paul**. From within this directory, you have to type **cd /home** instead of **cd home** to go to the **/home** directory.

```
paul@debian8$ pwd
/home/paul
paul@debian8$ cd home
bash: cd: home: No such file or directory
paul@debian8$ cd /home
paul@debian8$ pwd
/home
```

When inside **/home**, you have to type **cd paul** instead of **cd /paul** to enter the subdirectory **paul** of the current directory **/home**.

```
paul@debian8$ pwd
/home
paul@debian8$ cd /paul
bash: cd: /paul: No such file or directory
paul@debian8$ cd paul
paul@debian8$ pwd
/home/paul
```

In case your current directory is the **root directory** /, then both **cd /home** and **cd home** will get you in the **/home** directory.

```
paul@debian8$ pwd
/
paul@debian8$ cd home
paul@debian8$ pwd
/home
paul@debian8$ cd /
paul@debian8$ cd /home
paul@debian8$ pwd
/home
```

8.4. path completion

The **tab key** can help you in typing a path without errors. Typing **cd /et** followed by the **tab key** will expand the command line to **cd /etc/**. When typing **cd /Et** followed by the **tab key**, nothing will happen because you typed the wrong **path** (upper case E).

You will need fewer key strokes when using the **tab key**, and you will be sure your typed **path** is correct!

8.5. ls

You can list the contents of a directory with **ls**.

```
paul@debian8:~$ ls
allfiles.txt  dmesg.txt  services   stuff  summer.txt
paul@debian8:~$
```

8.5.1. ls -a

A frequently used option with ls is **-a** to show all files. Showing all files means including the **hidden files**. When a file name on a Linux file system starts with a dot, it is considered a **hidden file** and it doesn't show up in regular file listings.

```
paul@debian8:~$ ls
allfiles.txt  dmesg.txt  services  stuff  summer.txt
paul@debian8:~$ ls -a
.   allfiles.txt   .bash_profile  dmesg.txt   .lesshst  stuff
..  .bash_history  .bashrc        services    .ssh      summer.txt
paul@debian8:~$
```

8.5.2. ls -l

Many times you will be using options with **ls** to display the contents of the directory in different formats or to display different parts of the directory. Typing just **ls** gives you a list of files in the directory. Typing **ls -l** (that is a letter L, not the number 1) gives you a long listing.

```
paul@debian8:~$ ls -l
total 17296
-rw-r--r-- 1 paul paul 17584442 Sep 17 00:03 allfiles.txt
-rw-r--r-- 1 paul paul    96650 Sep 17 00:03 dmesg.txt
-rw-r--r-- 1 paul paul    19558 Sep 17 00:04 services
drwxr-xr-x 2 paul paul     4096 Sep 17 00:04 stuff
-rw-r--r-- 1 paul paul        0 Sep 17 00:04 summer.txt
```

8.5.3. ls -lh

ls -l -h ls -lh ls -hl ls -h -l

Another frequently used ls option is **-h**. It shows the numbers (file sizes) in a more human readable format. Also shown below is some variation in the way you can give the options to **ls**. We will explain the details of the output later in this book.

Note that we use the letter L as an option in this screenshot, not the number 1.

```
paul@debian8:~$ ls -l -h
total 17M
-rw-r--r-- 1 paul paul  17M Sep 17 00:03 allfiles.txt
-rw-r--r-- 1 paul paul  95K Sep 17 00:03 dmesg.txt
-rw-r--r-- 1 paul paul  20K Sep 17 00:04 services
drwxr-xr-x 2 paul paul 4.0K Sep 17 00:04 stuff
-rw-r--r-- 1 paul paul    0 Sep 17 00:04 summer.txt
```

8.6. mkdir

Walking around the Unix file tree is fun, but it is even more fun to create your own directories with **mkdir**. You have to give at least one parameter to **mkdir**, the name of the new directory to be created. Think before you type a leading / .

```
paul@debian8:~$ mkdir mydir
paul@debian8:~$ cd mydir
paul@debian8:~/mydir$ ls -al
total 8
drwxr-xr-x  2 paul paul 4096 Sep 17 00:07 .
drwxr-xr-x 48 paul paul 4096 Sep 17 00:07 ..
paul@debian8:~/mydir$ mkdir stuff
paul@debian8:~/mydir$ mkdir otherstuff
paul@debian8:~/mydir$ ls -l
total 8
drwxr-xr-x 2 paul paul 4096 Sep 17 00:08 otherstuff
drwxr-xr-x 2 paul paul 4096 Sep 17 00:08 stuff
paul@debian8:~/mydir$
```

8.6.1. mkdir -p

The following command will fail, because the **parent directory** of **threedirsdeep** does not exist.

```
paul@debian8:~$ mkdir mydir2/mysubdir2/threedirsdeep
mkdir: cannot create directory ‘mydir2/mysubdir2/threedirsdeep’: No such fi\
le or directory
```

When given the option **-p**, then **mkdir** will create **parent directories** as needed.

```
paul@debian8:~$ mkdir -p mydir2/mysubdir2/threedirsdeep
paul@debian8:~$ cd mydir2
paul@debian8:~/mydir2$ ls -l
total 4
drwxr-xr-x 3 paul paul 4096 Sep 17 00:11 mysubdir2
paul@debian8:~/mydir2$ cd mysubdir2
paul@debian8:~/mydir2/mysubdir2$ ls -l
total 4
drwxr-xr-x 2 paul paul 4096 Sep 17 00:11 threedirsdeep
paul@debian8:~/mydir2/mysubdir2$ cd threedirsdeep/
paul@debian8:~/mydir2/mysubdir2/threedirsdeep$ pwd
/home/paul/mydir2/mysubdir2/threedirsdeep
```

8.7. rmdir

When a directory is empty, you can use **rmdir** to remove the directory.

```
paul@debian8:~/mydir$ ls -l
total 8
drwxr-xr-x 2 paul paul 4096 Sep 17 00:08 otherstuff
drwxr-xr-x 2 paul paul 4096 Sep 17 00:08 stuff
paul@debian8:~/mydir$ rmdir otherstuff
paul@debian8:~/mydir$ cd ..
paul@debian8:~$ rmdir mydir
rmdir: failed to remove 'mydir': Directory not empty
paul@debian8:~$ rmdir mydir/stuff
paul@debian8:~$ rmdir mydir
paul@debian8:~$
```

8.7.1. rmdir -p

And similar to the **mkdir -p** option, you can also use **rmdir** to recursively remove directories.

```
paul@debian8:~$ mkdir -p test42/subdir
paul@debian8:~$ rmdir -p test42/subdir
paul@debian8:~$
```

working with files

In this chapter we learn how to recognise, create, remove, copy and move files using commands like **file, touch, rm, cp, mv** and **rename**.

9.1. all files are case sensitive

Files on Linux (or any Unix) are **case sensitive**. This means that **FILE1** is different from **file1**, and **/etc/hosts** is different from **/etc/Hosts** (the latter one does not exist on a typical Linux computer).

This screenshot shows the difference between two files, one with upper case **W**, the other with lower case **w**.

```
paul@laika:~/Linux$ ls
winter.txt  Winter.txt
paul@laika:~/Linux$ cat winter.txt
It is cold.
paul@laika:~/Linux$ cat Winter.txt
It is very cold!
```

9.2. everything is a file

A **directory** is a special kind of **file**, but it is still a (case sensitive!) **file**. Each terminal window (for example **/dev/pts/4**), any hard disk or partition (for example **/dev/sdb1**) and any process are all represented somewhere in the **file system** as a **file**. It will become clear throughout this course that everything on Linux is a **file**.

9.3. file

The **file** utility determines the file type. Linux does not use extensions to determine the file type. The command line does not care whether a file ends in .txt or .pdf. As a system administrator, you should use the **file** command to determine the file type. Here are some examples on a typical Linux system.

```
paul@laika:~$ file pic33.png
pic33.png: PNG image data, 3840 x 1200, 8-bit/color RGBA, non-interlaced
paul@laika:~$ file /etc/passwd
/etc/passwd: ASCII text
paul@laika:~$ file HelloWorld.c
HelloWorld.c: ASCII C program text
```

The file command uses a magic file that contains patterns to recognise file types. The magic file is located in **/usr/share/file/magic**. Type **man 5 magic** for more information.

It is interesting to point out **file -s** for special files like those in **/dev** and **/proc**.

```
root@debian6~# file /dev/sda
/dev/sda: block special
root@debian6~# file -s /dev/sda
/dev/sda: x86 boot sector; partition 1: ID=0x83, active, starthead...
root@debian6~# file /proc/cpuinfo
/proc/cpuinfo: empty
root@debian6~# file -s /proc/cpuinfo
/proc/cpuinfo: ASCII C++ program text
```

9.4. touch

9.4.1. create an empty file

One easy way to create an empty file is with **touch**. (We will see many other ways for creating files later in this book.)

This screenshot starts with an empty directory, creates two files with **touch** and the lists those files.

```
paul@debian7:~$ ls -l
total 0
paul@debian7:~$ touch file42
paul@debian7:~$ touch file33
paul@debian7:~$ ls -l
total 0
-rw-r--r-- 1 paul paul 0 Oct 15 08:57 file33
-rw-r--r-- 1 paul paul 0 Oct 15 08:56 file42
paul@debian7:~$
```

9.4.2. touch -t

The **touch** command can set some properties while creating empty files. Can you determine what is set by looking at the next screenshot? If not, check the manual for **touch**.

```
paul@debian7:~$ touch -t 200505050000 SinkoDeMayo
paul@debian7:~$ touch -t 130207111630 BigBattle.txt
paul@debian7:~$ ls -l
total 0
-rw-r--r-- 1 paul paul 0 Jul 11  1302 BigBattle.txt
-rw-r--r-- 1 paul paul 0 Oct 15 08:57 file33
-rw-r--r-- 1 paul paul 0 Oct 15 08:56 file42
-rw-r--r-- 1 paul paul 0 May  5  2005 SinkoDeMayo
paul@debian7:~$
```

9.5.1. remove forever

When you no longer need a file, use **rm** to remove it. Unlike some graphical user interfaces, the command line in general does not have a **waste bin** or **trash can** to recover files. When you use **rm** to remove a file, the file is gone. Therefore, be careful when removing files!

```
paul@debian7:~$ ls
BigBattle.txt  file33  file42  SinkoDeMayo
paul@debian7:~$ rm BigBattle.txt
paul@debian7:~$ ls
file33  file42  SinkoDeMayo
paul@debian7:~$
```

9.5.2. rm -i

To prevent yourself from accidentally removing a file, you can type **rm -i**.

```
paul@debian7:~$ ls
file33  file42  SinkoDeMayo
paul@debian7:~$ rm -i file33
rm: remove regular empty file `file33'? yes
paul@debian7:~$ rm -i SinkoDeMayo
rm: remove regular empty file `SinkoDeMayo'? n
paul@debian7:~$ ls
file42  SinkoDeMayo
paul@debian7:~$
```

9.5.3. rm -rf

By default, **rm -r** will not remove non-empty directories. However **rm** accepts several options that will allow you to remove any directory. The **rm -rf** statement is famous because it will erase anything (providing that you have the permissions to do so). When you are logged on as root, be very careful with **rm -rf** (the **f** means **force** and the **r** means **recursive**) since being root implies that permissions don't apply to you. You can literally erase your entire file system by accident.

```
paul@debian7:~$ mkdir test
paul@debian7:~$ rm test
rm: cannot remove `test': Is a directory
paul@debian7:~$ rm -rf test
paul@debian7:~$ ls test
ls: cannot access test: No such file or directory
paul@debian7:~$
```

9.6.1. copy one file

To copy a file, use **cp** with a source and a target argument.

```
paul@debian7:~$ ls
file42  SinkoDeMayo
paul@debian7:~$ cp file42 file42.copy
paul@debian7:~$ ls
file42  file42.copy  SinkoDeMayo
```

9.6.2. copy to another directory

If the target is a directory, then the source files are copied to that target directory.

```
paul@debian7:~$ mkdir dir42
paul@debian7:~$ cp SinkoDeMayo dir42
paul@debian7:~$ ls dir42/
SinkoDeMayo
```

9.6.3. cp -r

To copy complete directories, use **cp -r** (the **-r** option forces **recursive** copying of all files in all subdirectories).

```
paul@debian7:~$ ls
dir42  file42  file42.copy  SinkoDeMayo
paul@debian7:~$ cp -r dir42/ dir33
paul@debian7:~$ ls
dir33  dir42  file42  file42.copy  SinkoDeMayo
paul@debian7:~$ ls dir33/
SinkoDeMayo
```

9.6.4. copy multiple files to directory

You can also use cp to copy multiple files into a directory. In this case, the last argument (a.k.a. the target) must be a directory.

```
paul@debian7:~$ cp file42 file42.copy SinkoDeMayo dir42/
paul@debian7:~$ ls dir42/
file42  file42.copy  SinkoDeMayo
```

9.6.5. cp -i

To prevent **cp** from overwriting existing files, use the **-i** (for interactive) option.

```
paul@debian7:~$ cp SinkoDeMayo file42
paul@debian7:~$ cp SinkoDeMayo file42
paul@debian7:~$ cp -i SinkoDeMayo file42
cp: overwrite `file42'? n
paul@debian7:~$
```

9.7.1. rename files with mv

Use **mv** to rename a file or to move the file to another directory.

```
paul@debian7:~$ ls
dir33  dir42  file42  file42.copy  SinkoDeMayo
paul@debian7:~$ mv file42 file33
paul@debian7:~$ ls
dir33  dir42  file33  file42.copy  SinkoDeMayo
paul@debian7:~$
```

When you need to rename only one file then **mv** is the preferred command to use.

9.7.2. rename directories with mv

The same **mv** command can be used to rename directories.

```
paul@debian7:~$ ls -l
total 8
drwxr-xr-x 2 paul paul 4096 Oct 15 09:36 dir33
drwxr-xr-x 2 paul paul 4096 Oct 15 09:36 dir42
-rw-r--r-- 1 paul paul    0 Oct 15 09:38 file33
-rw-r--r-- 1 paul paul    0 Oct 15 09:16 file42.copy
-rw-r--r-- 1 paul paul    0 May  5  2005 SinkoDeMayo
paul@debian7:~$ mv dir33 backup
paul@debian7:~$ ls -l
total 8
drwxr-xr-x 2 paul paul 4096 Oct 15 09:36 backup
drwxr-xr-x 2 paul paul 4096 Oct 15 09:36 dir42
-rw-r--r-- 1 paul paul    0 Oct 15 09:38 file33
-rw-r--r-- 1 paul paul    0 Oct 15 09:16 file42.copy
-rw-r--r-- 1 paul paul    0 May  5  2005 SinkoDeMayo
paul@debian7:~$
```

9.7.3. mv -i

The **mv** also has a **-i** switch similar to **cp** and **rm**.

this screenshot shows that **mv -i** will ask permission to overwrite an existing file.

```
paul@debian7:~$ mv -i file33 SinkoDeMayo
mv: overwrite `SinkoDeMayo'? no
paul@debian7:~$
```

working with file contents

In this chapter we will look at the contents of **text files** with **head, tail, cat, tac, more, less** and **strings**.

We will also get a glimpse of the possibilities of tools like **cat** on the command line.

10.1. head

You can use **head** to display the first ten lines of a file.

```
paul@debian7~$ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
root@debian7~#
```

The **head** command can also display the first **n** lines of a file.

```
paul@debian7~$ head -4 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
paul@debian7~$
```

And **head** can also display the first **n bytes**.

```
paul@debian7~$ head -c14 /etc/passwd
root:x:0:0:roopaul@debian7~$
```

10.2. tail

Similar to **head**, the **tail** command will display the last ten lines of a file.

```
paul@debian7~$ tail /etc/services
vboxd           20012/udp
binkp           24554/tcp                       # binkp fidonet protocol
asp             27374/tcp                       # Address Search Protocol
asp             27374/udp
csync2          30865/tcp                       # cluster synchronization tool
dircproxy       57000/tcp                       # Detachable IRC Proxy
tfido           60177/tcp                       # fidonet EMSI over telnet
fido            60179/tcp                       # fidonet EMSI over TCP

# Local services
paul@debian7~$
```

You can give **tail** the number of lines you want to see.

```
paul@debian7~$ tail -3 /etc/services
fido            60179/tcp                       # fidonet EMSI over TCP

# Local services
paul@debian7~$
```

10.3. cat

The **cat** command is one of the most universal tools, yet all it does is copy **standard input** to **standard output**. In combination with the shell this can be very powerful and diverse. Some examples will give a glimpse into the possibilities. The first example is simple, you can use cat to display a file on the screen. If the file is longer than the screen, it will scroll to the end.

```
paul@debian8:~$ cat /etc/resolv.conf
domain linux-training.be
search linux-training.be
nameserver 192.168.1.42
```

10.3.1. concatenate

cat is short for **concatenate**. One of the basic uses of **cat** is to concatenate files into a bigger (or complete) file.

```
paul@debian8:~$ echo one >part1
paul@debian8:~$ echo two >part2
paul@debian8:~$ echo three >part3
paul@debian8:~$ cat part1
one
paul@debian8:~$ cat part2
two
paul@debian8:~$ cat part3
three
paul@debian8:~$ cat part1 part2 part3
one
two
three
paul@debian8:~$ cat part1 part2 part3 >all
paul@debian8:~$ cat all
one
two
three
paul@debian8:~$
```

10.3.2. create files

You can use **cat** to create flat text files. Type the **cat > winter.txt** command as shown in the screenshot below. Then type one or more lines, finishing each line with the enter key. After the last line, type and hold the Control (Ctrl) key and press d.

```
paul@debian8:~$ cat > winter.txt
It is very cold today!
paul@debian8:~$ cat winter.txt
It is very cold today!
paul@debian8:~$
```

The **Ctrl d** key combination will send an **EOF** (End of File) to the running process ending the **cat** command.

10.3.4. copy files

In the third example you will see that cat can be used to copy files. We will explain in detail what happens here in the bash shell chapter.

```
paul@debian8:~$ cat winter.txt
It is very cold today!
paul@debian8:~$ cat winter.txt > cold.txt
paul@debian8:~$ cat cold.txt
It is very cold today!
paul@debian8:~$
```

10.4. tac

Just one example will show you the purpose of **tac** (cat backwards).

```
paul@debian8:~$ cat count
one
two
three
four
paul@debian8:~$ tac count
four
three
two
one
```

```
[paul@RHEL4b ~]$ echo 'A line with      single    quotes'
A line with      single    quotes
[paul@RHEL4b ~]$

[paul@RHEL4b ~]$ echo "A line with      double    quotes"
A line with      double    quotes
[paul@RHEL4b ~]$
```

Quoted lines can include special escaped characters recognised by the **echo** command (when using **echo -e**). The screenshot below shows how to use **\n** for a newline and **\t** for a tab (usually eight white spaces).

```
[paul@RHEL4b ~]$ echo -e "A line with \na newline"
A line with
a newline
[paul@RHEL4b ~]$ echo -e 'A line with \na newline'
A line with
a newline
[paul@RHEL4b ~]$ echo -e "A line with \ta tab"
A line with     a tab
[paul@RHEL4b ~]$ echo -e 'A line with \ta tab'
A line with     a tab
[paul@RHEL4b ~]$
```

type

To find out whether a command given to the shell will be executed as an **external command** or as a **builtin command**, use the **type** command.

```
paul@laika:~$ type cd
cd is a shell builtin
paul@laika:~$ type cat
cat is /bin/cat
paul@laika:~$ type ls
ls is aliased to `ls --color=auto'
```

12.6.3. running external commands

Some commands have both builtin and external versions. When one of these commands is executed, the builtin version takes priority. To run the external version, you must enter the full path to the command.

```
paul@laika:~$ type -a echo
echo is a shell builtin
echo is /bin/echo
paul@laika:~$ /bin/echo Running the external echo command...
Running the external echo command...
```

12.6.4. which

The **which** command will search for binaries in the **$PATH** environment variable (variables will be explained later). In the screenshot below, it is determined that **cd** is **builtin**, and **ls, cp, rm, mv, mkdir, pwd,** and **which** are external commands.

```
[root@RHEL4b ~]# which cp ls cd mkdir pwd
/bin/cp
/bin/ls
/usr/bin/which: no cd in (/usr/kerberos/sbin:/usr/kerberos/bin:...
/bin/mkdir
/bin/pwd
```

12.7.1. create an alias

The shell allows you to create **aliases**. Aliases are often used to create an easier to remember name for an existing command or to easily supply parameters.

```
[paul@RHELv4u3 ~]$ cat count.txt
one
two
three
[paul@RHELv4u3 ~]$ alias dog=tac
[paul@RHELv4u3 ~]$ dog count.txt
three
two
one
```

12.7.2. abbreviate commands

An **alias** can also be useful to abbreviate an existing command.

```
paul@laika:~$ alias ll='ls -lh --color=auto'
paul@laika:~$ alias c='clear'
paul@laika:~$
```

12.7.3. default options

Aliases can be used to supply commands with default options. The example below shows how to set the **-i** option default when typing **rm**.

```
[paul@RHELv4u3 ~]$ rm -i winter.txt
rm: remove regular file `winter.txt'? no
[paul@RHELv4u3 ~]$ rm winter.txt
[paul@RHELv4u3 ~]$ ls winter.txt
ls: winter.txt: No such file or directory
[paul@RHELv4u3 ~]$ touch winter.txt
[paul@RHELv4u3 ~]$ alias rm='rm -i'
[paul@RHELv4u3 ~]$ rm winter.txt
rm: remove regular empty file `winter.txt'? no
[paul@RHELv4u3 ~]$
```

Some distributions enable default aliases to protect users from accidentally erasing files ('rm -i', 'mv -i', 'cp -i')

12.7.5. unalias

You can undo an alias with the **unalias** command.

```
[paul@RHEL4b ~]$ which rm
/bin/rm
[paul@RHEL4b ~]$ alias rm='rm -i'
[paul@RHEL4b ~]$ which rm
alias rm='rm -i'
        /bin/rm
[paul@RHEL4b ~]$ unalias rm
[paul@RHEL4b ~]$ which rm
/bin/rm
[paul@RHEL4b ~]$
```

12.8. displaying shell expansion

You can display shell expansion with **set -x**, and stop displaying it with **set +x**. You might want to use this further on in this course, or when in doubt about exactly what the shell is doing with your command.

```
[paul@RHELv4u3 ~]$ set -x
++ echo -ne '\033]0;paul@RHELv4u3:~\007'
[paul@RHELv4u3 ~]$ echo $USER
+ echo paul
paul
++ echo -ne '\033]0;paul@RHELv4u3:~\007'
[paul@RHELv4u3 ~]$ echo \$USER
+ echo '$USER'
$USER
++ echo -ne '\033]0;paul@RHELv4u3:~\007'
[paul@RHELv4u3 ~]$ set +x
+ set +x
[paul@RHELv4u3 ~]$ echo $USER
paul
```

Control Operators

; semicolon

All the commands will be executed sequentially waiting for each command to finish before starting the new one

```
[paul@RHELv4u3 ~]$ echo Hello
Hello
[paul@RHELv4u3 ~]$ echo World
World
[paul@RHELv4u3 ~]$ echo Hello ; echo World
Hello
World
[paul@RHELv4u3 ~]$
```

& ampersand (background execution)

```
[paul@RHELv4u3 ~]$ sleep 20 &
[1] 7925
[paul@RHELv4u3 ~]$
...wait 20 seconds...
[paul@RHELv4u3 ~]$
[1]+ Done

sleep 20
```

&& double ampersand

```
paul@barry:~$ echo first && echo second
first
second
paul@barry:~$ zecho first && echo second
-bash: zecho: command not found
```

|| double vertical bar

The || represents a **logical OR**. The second command is executed only when the first command fails (returns a non-zero exit status).

```
paul@barry:~$ echo first || echo second ; echo third
first
third
paul@barry:~$ zecho first || echo second ; echo third
-bash: zecho: command not found
second
third
paul@barry:~$
```


```
[paul@RHELv4u3 ~]$ cd gen && ls
file1  file3  File55  fileab  FileAB   fileabc
file2  File4  FileA   Fileab  fileab2
[paul@RHELv4u3 gen]$ cd gen && ls
-bash: cd: gen: No such file or directory
```

$? dollar question mark

exit code of previous command

```
paul@debian5:~/test$ touch file1
paul@debian5:~/test$ echo $?
0
paul@debian5:~/test$ rm file1
paul@debian5:~/test$ echo $?
0
paul@debian5:~/test$ rm file1
rm: cannot remove `file1': No such file or directory
paul@debian5:~/test$ echo $?
1
paul@debian5:~/test$
```

Success

Error

```
paul@debian4:~$ mkdir test # we create a directory
paul@debian4:~/test$
```

You can use this logical AND and logical OR to write an **if-then-else** structure on the command line. This example uses **echo** to display whether the **rm** command was successful.

```
paul@laika:~/test$ rm file1 && echo It worked! || echo It failed!
It worked!
paul@laika:~/test$ rm file1 && echo It worked! || echo It failed!
rm: cannot remove `file1': No such file or directory
It failed!
paul@laika:~/test$
```

Escape '\\' and split '\'

```
[paul@RHELv4u3 ~]$ echo escaping \\\ \#\ \&\ \"\ \'
escaping \ # & " '
```

```
[paul@RHEL4b ~]$ echo This command line \
> is split in three \
> parts
This command line is split in three parts
[paul@RHEL4b ~]$
```

shell variables

$ dollar sign

```
[paul@RHELv4u3 ~]$ echo This is the $SHELL shell
This is the /bin/bash shell
[paul@RHELv4u3 ~]$ echo This is $SHELL on computer $HOSTNAME
This is /bin/bash on computer RHELv4u3.localdomain
[paul@RHELv4u3 ~]$ echo The userid of $USER is $UID
The userid of paul is 500
[paul@RHELv4u3 ~]$ echo My homedir is $HOME
My homedir is /home/paul
```

```
[paul@RHELv4u3 ~]$ echo Hello $USER
Hello paul
[paul@RHELv4u3 ~]$ echo Hello $user
Hello
[paul@RHELv4u3 gen]$ MyVar=555
[paul@RHELv4u3 gen]$ echo $MyVar
555
[paul@RHELv4u3 gen]$
```

```
[paul@RHELv4u3 ~]$ MyVar=555
[paul@RHELv4u3 ~]$ echo $MyVar
555
[paul@RHELv4u3 ~]$ echo "$MyVar"
555
[paul@RHELv4u3 ~]$ echo '$MyVar'
$MyVar
```

!

```
[paul@RHEL4b ~]$ MyVar=8472
[paul@RHEL4b ~]$ echo $MyVar
8472
[paul@RHEL4b ~]$ unset MyVar
[paul@RHEL4b ~]$ echo $MyVar

[paul@RHEL4b ~]$
```

$PATH

The **$PATH** variable is determines where the shell is looking for commands to execute (unless the command is builtin or aliased). This variable contains a list of directories, separated by colons.

```
[[paul@RHEL4b ~]$ echo $PATH
/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:
```

```
[paul@RHEL4b ~]$ prefix=Super
[paul@RHEL4b ~]$ echo Hello $prefixman and $prefixgirl
Hello  and
[paul@RHEL4b ~]$ echo Hello ${prefix}man and ${prefix}girl
Hello Superman and Supergirl
[paul@RHEL4b ~]$
```

katholieke hogeschool associatie KU Leuven vives

output redirection

```
[paul@RHELv4u3 ~]$ echo It is cold today!
It is cold today!
[paul@RHELv4u3 ~]$ echo It is cold today! > winter.txt
[paul@RHELv4u3 ~]$ cat winter.txt
It is cold today!
[paul@RHELv4u3 ~]$
```

```
[paul@RHELv4u3 ~]$ cat winter.txt
It is cold today!
[paul@RHELv4u3 ~]$ zcho It is cold today! > winter.txt
-bash: zcho: command not found
[paul@RHELv4u3 ~]$ cat winter.txt
[paul@RHELv4u3 ~]$
```

2> stderr

```
cat winter.txt > snow.txt 2> errors.txt
```

>> append

```
[paul@RHELv4u3 ~]$ echo It is cold today! > winter.txt
[paul@RHELv4u3 ~]$ cat winter.txt
It is cold today!
[paul@RHELv4u3 ~]$ echo Where is the summer ? >> winter.txt
[paul@RHELv4u3 ~]$ cat winter.txt
It is cold today!
Where is the summer ?
[paul@RHELv4u3 ~]$
```

grep

```
[paul@RHEL4b pipes]$ cat tennis.txt
Amelie Mauresmo, Fra
Kim Clijsters, BEL
Justine Henin, Bel
Serena Williams, usa
Venus Williams, USA
[paul@RHEL4b pipes]$ cat tennis.txt | grep Williams
Serena Williams, usa
Venus Williams, USA
```

The most common use of **grep** is to filter lines of text containing (or not containing) a certain string.

One of the most useful options of grep is **grep -i** which filters in a case insensitive way

filters

Commands that are created to be used with a **pipe** are often called **filters**. These **filters** are very small programs that do one specific thing very efficiently. They can be used as **building blocks**.

```
[paul@RHEL4b pipes]$ tac count.txt | cat | cat |
cat | cat | cat
five
four
three
two
one
[paul@RHEL4b pipes]$
```

```
[paul@RHEL4b pipes]$ tac count.txt | tee temp.txt | tac
one
two
three
four
five
[paul@RHEL4b pipes]$ cat temp.txt
five
four
three
two
one
[paul@RHEL4b pipes]$
```

```
grep Williams tenni
```

Another very useful option is **grep -v** which outputs lines not matching the string.

```
[paul@RHEL4b pipes]$ grep -v Fra tennis.txt
Kim Clijsters, BEL
Justine Henin, Bel
Serena Williams, usa
Venus Williams, USA
[paul@RHEL4b pipes]$
```

And of course, both options can be combined to filter all lines not containing a case insensitive string.

```
[paul@RHEL4b pipes]$ grep -vi usa tennis.txt
Amelie Mauresmo, Fra
Kim Clijsters, BEL
Justine Henin, Bel
[paul@RHEL4b pipes]$
```

With **grep -A1** one line **after** the result is also displayed.

```
paul@debian5:~/pipes$ grep -A1 Henin tennis.txt
Justine Henin, Bel
Serena Williams, usa
```

With **grep -B1** one line **before** the result is also displayed.

```
paul@debian5:~/pipes$ grep -B1 Henin tennis.txt
Kim Clijsters, BEL
Justine Henin, Bel
```

With **grep -C1** (context) one line **before** and one **after** are also displayed. All three options (A,B, and C) can display any number of lines (using e.g. A2, B4 or C20).

```
paul@debian5:~/pipes$ grep -C1 Henin tennis.txt
Kim Clijsters, BEL
Justine Henin, Bel
Serena Williams, usa
```

cut

The **cut** filter can select columns from files, depending on a delimiter or a count of bytes. The screenshot below uses **cut** to filter for the username and userid in the **/etc/passwd** file. It uses the colon as a delimiter, and selects fields 1 and 3.

```
[[paul@RHEL4b pipes]$ cut -d: -f1,3 /etc/passwd | tail -4
Figo:510
Pfaff:511
Harry:516
Hermione:517
[paul@RHEL4b pipes]$
```

wc

Counting words, lines and characters is easy with **wc**.

```
paul@RHEL4b pipes]$ wc tennis.txt
  5  15 100 tennis.txt
[paul@RHEL4b pipes]$ wc -l tennis.txt
5 tennis.txt
[paul@RHEL4b pipes]$ wc -w tennis.txt
15 tennis.txt
[paul@RHEL4b pipes]$ wc -c tennis.txt
100 tennis.txt
[paul@RHEL4b pipes]$
```

uniq

With **uniq** you can remove duplicates from a **sorted list**.

```
paul@debian5:~/pipes$ cat music.txt
Queen
Brel
Queen
Abba
paul@debian5:~/pipes$ sort music.txt
Abba
Brel
Queen
Queen
paul@debian5:~/pipes$ sort music.txt |uniq
Abba
Brel
Queen
```

```
paul@debian5:~/pipes$ sort music.txt |uniq -c
      1 Abba
1 Brel 2 Queen
```

sort

```
paul@debian5:~/pipes$ cat music.txt
Queen
Brel
Led Zeppelin
Abba
paul@debian5:~/pipes$ sort music.txt
Abba
Brel
Led Zeppelin
Queen
```

```
[paul@RHEL4b pipes]$ sort -k1 country.txt
Belgium, Brussels, 10
France, Paris, 60
Germany, Berlin, 100
Iran, Teheran, 70
Italy, Rome, 50
[paul@RHEL4b pipes]$ sort -k2 country.txt
Germany, Berlin, 100
Belgium, Brussels, 10
France, Paris, 60
Italy, Rome, 50
Iran, Teheran, 70
```

comm

Comparing streams (or files) can be done with the **comm**. By default **comm** will output three columns. In this example, Abba, Cure and Queen are in both lists, Bowie and Sweet are only in the first file, Turner is only in the second.

```
paul@debian5:~/pipes$ cat > list1.txt
Abba
Bowie
Cure
Queen
Sweet
paul@debian5:~/pipes$ cat > list2.txt
Abba
Cure
Queen
Turner
paul@debian5:~/pipes$ comm list1.txt list2.txt
                    Abba
Bowie
                    Cure
                    Queen
Sweet
        Turner
```

19.11. sed

The stream **ed**itor **sed** can perform editing functions in the stream, using **regular expressions**.

```
paul@debian5:~/pipes$ echo level5 | sed 's/5/42/'
level42
paul@debian5:~/pipes$ echo level5 | sed 's/level/jump/'
jump5
```

Add **g** for global replacements (all occurrences of the string per line).

```
paul@debian5:~/pipes$ echo level5 level7 | sed 's/level/jump/'
jump5 level7
paul@debian5:~/pipes$ echo level5 level7 | sed 's/level/jump/g'
jump5 jump7
```

With **d** you can remove lines from a stream containing a character.

```
paul@debian5:~/test42$ cat tennis.txt
Venus Williams, USA
Martina Hingis, SUI
Justine Henin, BE
Serena williams, USA
Kim Clijsters, BE
Yanina Wickmayer, BE
paul@debian5:~/test42$ cat tennis.txt | sed '/BE/d'
Venus Williams, USA
Martina Hingis, SUI
Serena williams, USA
```

19.12. pipe examples

19.12.1. who | wc

How many users are logged on to this system ?

```
[paul@RHEL4b pipes]$ who
root     tty1         Jul 25 10:50
paul     pts/0        Jul 25 09:29 (laika)
Harry    pts/1        Jul 25 12:26 (barry)
paul     pts/2        Jul 25 12:26 (pasha)
[paul@RHEL4b pipes]$ who | wc -l
4
```

19.12.2. who | cut | sort

Display a sorted list of logged on users.

```
[paul@RHEL4b pipes]$ who | cut -d' ' -f1 | sort
Harry
paul
paul
root
```

Display a sorted list of logged on users, but every user only once .

```
[paul@RHEL4b pipes]$ who | cut -d' ' -f1 | sort | uniq
Harry
paul
root
```

19.12.3. grep | cut

Display a list of all bash **user accounts** on this computer. Users accounts are explained in detail later.

```
paul@debian5:~$ grep bash /etc/passwd
root:x:0:0:root:/root:/bin/bash
paul:x:1000:1000:paul,,,:/home/paul:/bin/bash
serena:x:1001:1001::/home/serena:/bin/bash
paul@debian5:~$ grep bash /etc/passwd | cut -d: -f1
root
paul
serena
```

20.1. find

The **find** command can be very useful at the start of a pipe to search for files. Here are some examples. You might want to add **2>/dev/null** to the command lines to avoid cluttering your screen with error messages.

Find all files in **/etc** and put the list in etcfiles.txt

```
find /etc > etcfiles.txt
```

Find all files of the entire system and put the list in allfiles.txt

```
find / > allfiles.txt
```

Find files that end in .conf in the current directory (and all subdirs).

```
find . -name "*.conf"
```

Find files of type file (not directory, pipe or etc.) that end in .conf.

```
find . -type f -name "*.conf"
```

Find files of type directory that end in .bak .

```
find /data -type d -name "*.bak"
```

Find files that are newer than file42.txt

```
find . -newer file42.txt
```

Find can also execute another command on every file found. This example will look for *.odf files and copy them to /backup/.

```
find /data -name "*.odf" -exec cp {} /backup/ \;
```

Find can also execute, after your confirmation, another command on every file found. This example will remove *.odf files if you approve of it for every file found.

```
find /data -name "*.odf" -ok rm {} \;
```

20.3. date

The **date** command can display the date, time, time zone and more.

```
paul@rhel55 ~$ date
Sat Apr 17 12:44:30 CEST 2010
```

A date string can be customised to display the format of your choice. Check the man page for more options.

```
paul@rhel55 ~$ date +'%A %d-%m-%Y'
Saturday 17-04-2010
```

Time on any Unix is calculated in number of seconds since 1969 (the first second being the first second of the first of January 1970). Use **date +%s** to display Unix time in seconds.

```
paul@rhel55 ~$ date +%s
1271501080
```

When will this seconds counter reach two thousand million ?

```
paul@rhel55 ~$ date -d '1970-01-01 + 2000000000 seconds'
Wed May 18 04:33:20 CEST 2033
```

20.4. cal

The **cal** command displays the current month, with the current day highlighted.

```
paul@rhel55 ~$ cal
     April 2010
Su Mo Tu We Th Fr Sa
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30
```

You can select any month in the past or the future.

```
paul@rhel55 ~$ cal 2 1970
   February 1970
Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
```

20.5. sleep

The **sleep** command is sometimes used in scripts to wait a number of seconds. This example shows a five second **sleep**.

```
paul@rhel55 ~$ sleep 5
paul@rhel55 ~$
```

20.6. time

The **time** command can display how long it takes to execute a command. The **date** command takes only a little time.

```
paul@rhel55 ~$ time date
Sat Apr 17 13:08:27 CEST 2010

real    0m0.014s
user    0m0.008s
sys     0m0.006s
```

The **sleep 5** command takes five **real** seconds to execute, but consumes little **cpu time**.

```
paul@rhel55 ~$ time sleep 5

real    0m5.018s
user    0m0.005s
sys     0m0.011s
```

This **bzip2** command compresses a file and uses a lot of **cpu time**.

```
paul@rhel55 ~$ time bzip2 text.txt

real    0m2.368s
user    0m0.847s
sys     0m0.539s
```

20.7. gzip - gunzip

bzip2 - bunzip2

Users never have enough disk space, so compression comes in handy. The **gzip** command can make files take up less space.

```
paul@rhel55 ~$ ls -lh text.txt
-rw-rw-r-- 1 paul paul 6.4M Apr 17 13:11 text.txt
paul@rhel55 ~$ gzip text.txt
paul@rhel55 ~$ ls -lh text.txt.gz
-rw-rw-r-- 1 paul paul 760K Apr 17 13:11 text.txt.gz
```

You can get the original back with **gunzip**.

```
paul@rhel55 ~$ gunzip text.txt.gz
paul@rhel55 ~$ ls -lh text.txt
-rw-rw-r-- 1 paul paul 6.4M Apr 17 13:11 text.txt
```

20.8. zcat - zmore

bzcat - bzmore

Text files that are compressed with **gzip** can be viewed with **zcat** and **zmore**.

```
paul@rhel55 ~$ head -4 text.txt
/
/opt
/opt/VBoxGuestAdditions-3.1.6
/opt/VBoxGuestAdditions-3.1.6/routines.sh
paul@rhel55 ~$ gzip text.txt
paul@rhel55 ~$ zcat text.txt.gz | head -4
/
/opt
/opt/VBoxGuestAdditions-3.1.6
/opt/VBoxGuestAdditions-3.1.6/routines.sh
```

21.1. regex versions

There are three different versions of regular expression syntax:

```
BRE: Basic Regular Expressions
ERE: Extended Regular Expressions
PRCE: Perl Regular Expressions
```

Depending on the tool being used, one or more of these syntaxes can be used.

For example the **grep** tool has the **-E** option to force a string to be read as ERE while **-G** forces BRE and **-P** forces PRCE.

Note that **grep** also has **-F** to force the string to be read literally.

The **sed** tool also has options to choose a regex syntax.

Read the manual of the tools you use!

```
paul@rhel65:~$ cat names
Tania
Laura
Valentina
```

When **grepping** for a single character, only the lines containing that character are returned.

```
paul@rhel65:~$ grep u names
Laura
paul@rhel65:~$ grep e names
Valentina
paul@rhel65:~$ grep i names
Tania
Valentina
```

```
paul@rhel65:~$ grep a names
Tania
Laura
Valentina
paul@rhel65:~$ grep ia names
Tania
paul@rhel65:~$ grep in names
Valentina
paul@rhel65:~$
```

21.2.3. one or the other

PRCE and ERE both use the pipe symbol to signify OR. In this example we **grep** for lines containing the letter i or the letter a.

```
paul@debian7:~$ cat list
Tania
Laura
paul@debian7:~$ grep -E 'i|a' list
Tania
Laura
```

Note that we use the **-E** switch of grep to force interpretion of our string as an ERE.

We need to **escape** the pipe symbol in a BRE to get the same logical OR.

```
paul@debian7:~$ grep -G 'i|a' list
paul@debian7:~$ grep -G 'i\|a' list
Tania
Laura
```

21.2.4. one or more

The * signifies zero, one or more occurences of the previous and the **+** signifies one or more of the previous.

```
paul@debian7:~$ cat list2
ll
lol
lool
loool
paul@debian7:~$ grep -E 'o*' list2
ll
lol
lool
loool
paul@debian7:~$ grep -E 'o+' list2
lol
lool
loool
paul@debian7:~$
```

21.2.5. match the end of a string

For the following examples, we will use this file.

```
paul@debian7:~$ cat names
Tania
Laura
Valentina
Fleur
Floor
```

The two examples below show how to use the **dollar character** to match the end of a string.

```
paul@debian7:~$ grep a$ names
Tania
Laura
Valentina
paul@debian7:~$ grep r$ names
Fleur
Floor
```

21.2.6. match the start of a string

The **caret character (^)** will match a string at the start (or the beginning) of a line.

Given the same file as above, here are two examples.

```
paul@debian7:~$ grep ^Val names
Valentina
paul@debian7:~$ grep ^F names
Fleur
Floor
```

Both the dollar sign and the little hat are called **anchors** in a regex.

21.2.7. separating words

Regular expressions use a **\b** sequence to reference a word separator. Take for example this file:

```
paul@debian7:~$ cat text
The governer is governing.
The winter is over.
Can you get over there?
```

Simply grepping for **over** will give too many results.

```
paul@debian7:~$ grep over text
The governer is governing.
The winter is over.
Can you get over there?
```

Surrounding the searched word with spaces is not a good solution (because other characters can be word separators). This screenshot below show how to use **\b** to find only the searched word:

```
paul@debian7:~$ grep '\bover\b' text
The winter is over.
Can you get over there?
paul@debian7:~$
```

Note that **grep** also has a **-w** option to grep for words.

```
paul@debian7:~$ cat text
The governer is governing.
The winter is over.
Can you get over there?
paul@debian7:~$ grep -w over text
The winter is over.
Can you get over there?
paul@debian7:~$
```

21.4. sed

21.4.1. stream editor

The **stream editor** or short **sed** uses **regex** for stream editing.

In this example **sed** is used to replace a string.

```
echo Sunday | sed 's/Sun/Mon/'
Monday
```

The slashes can be replaced by a couple of other characters, which can be handy in some cases to improve readability.

```
echo Sunday | sed 's:Sun:Mon:'
Monday
echo Sunday | sed 's_Sun_Mon_'
Monday
echo Sunday | sed 's|Sun|Mon|'
Monday
```

21.4.2. interactive editor

While **sed** is meant to be used in a stream, it can also be used interactively on a file.

```
paul@debian7:~/files$ echo Sunday > today
paul@debian7:~/files$ cat today
Sunday
paul@debian7:~/files$ sed -i 's/Sun/Mon/' today
paul@debian7:~/files$ cat today
Monday
```

21.4.9. exactly n times

You can demand an exact number of times the oprevious has to occur.

This example wants exactly three o's.

```
paul@debian7:~$ cat list2
ll
lol
lool
loool
paul@debian7:~$ grep -E 'o{3}' list2
loool
paul@debian7:~$ cat list2 | sed 's/o\{3\}/A/'
ll
lol
lool
lAl
paul@debian7:~$
```

21.4.10. between n and m times

And here we demand exactly from minimum 2 to maximum 3 times.

```
paul@debian7:~$ cat list2
ll
lol
lool
loool
paul@debian7:~$ grep -E 'o{2,3}' list2
lool
loool
paul@debian7:~$ grep 'o\{2,3\}' list2
lool
loool
paul@debian7:~$ cat list2 | sed 's/o\{2,3\}/A/'
ll
lol
lAl
lAl
paul@debian7:~$
```

```
paul@debian7:~$ cat list2
ll
lol
lool
loool
paul@debian7:~$ grep -E 'ooo?' list2
lool
loool
paul@debian7:~$ cat list2 | sed 's/ooo\?/A/'
ll
lol
lAl
lAl
```

21.5. bash history

The **bash shell** can also interprete some regular expressions.

This example shows how to manipulate the exclamation mask history feature of the bash shell.

```
paul@debian7:~$ mkdir hist
paul@debian7:~$ cd hist/
paul@debian7:~/hist$ touch file1 file2 file3
paul@debian7:~/hist$ ls -l file1
-rw-r--r-- 1 paul paul 0 Apr 15 22:07 file1
paul@debian7:~/hist$ !l
ls -l file1
-rw-r--r-- 1 paul paul 0 Apr 15 22:07 file1
paul@debian7:~/hist$ !l:s/1/3
ls -l file3
-rw-r--r-- 1 paul paul 0 Apr 15 22:07 file3
paul@debian7:~/hist$
```

This also works with the history numbers in bash.

```
paul@debian7:~/hist$ history 6
 2089  mkdir hist
 2090  cd hist/
 2091  touch file1 file2 file3
 2092  ls -l file1
 2093  ls -l file3
 2094  history 6
paul@debian7:~/hist$ !2092
ls -l file1
-rw-r--r-- 1 paul paul 0 Apr 15 22:07 file1
paul@debian7:~/hist$ !2092:s/1/2
ls -l file2
-rw-r--r-- 1 paul paul 0 Apr 15 22:07 file2
paul@debian7:~/hist$
```

Belangrijkste omgevingsvariabelen:

• $HOME	Home map van de gebruiker
• $USER / $USERNAME	Naam van de gebruiker
• $PWD	Huidige werkmap
• $BASH / $SHELL	Locatie van bash of shell
• $BASH_VERSION	Versie van Bash
• $OSTYPE	Info over het OS
• $PATH	Alle locaties voor uitvoerbare bestanden
• $IFS	Bevat alle scheidingsteken

Belangrijkste shellvariabelen:

- $0 scriptnaam
- $# Aantal meegegeven parameters
- $@ Alle meegegeven paramters
- $1, $2, … ${x} Positionele parameter
  x is getal voor laatste positie
- $? Uitvoerstatus laatste commando
- $$ Proces-id van het script
- $! Proces-id van het laatste commando opgestart met &

Practice 8.8 Blz 81 (13) - Working with directories
Practice 9.9 Blz 91 (21) - Working with files
Practice 10.7 Blz 99 (27) - File contents
Practice 11.10 Blz 118 (45) - File system tree
Practice 12.9 Blz 131 (56) - Commands and arguments
Practice 13.9 Blz 139 (62) - Control operators
Practice 19.13 Blz 192 (108) - Filters
Practice 20.11 Blz 202 (116) - Basic Unix tools

BONUS

Creër een scriptfile mijnsysteem.sh zodanig dat bij het runnen je de volgende output krijgt:

De script kan je laten uitvoeren aan de hand van het commando

bash mijnsysteem.

```
Deze informatie wordt aangeboden door de mijnsysteem.sh.
Het programma start nu!

Hallo,

Vandaag zijn we vr apr 16 12:44:57 CEST 2010, en dit is week 15.

Volgende gebruikers zijn momenteel aangemeld:

nathalie

Dit is Linux die draait op een i686 processor.

Ziehier de uptime informatie:
 12:44:57 up  2:59,  3 users,  load average: 0.00, 0.00, 0.00
```

Speciale karakters

\	Escape karakter
/	Scheidingsteken binnen een pad
.	Huidige directory
..	Parent directory
~	Home directory van de gebruiker
*	0 of meer karakters in een bestandsnaam, of op zichzelf, alle bestanden in een directory. Bijvoorbeeld: `pic*2012` kan staan voor pic2012, picJanuary2012, …
?	1 karakter in een bestandsnaam. Bijvoorbeeld: `hello?.txt` kan staan voor hello1.txt, hello2.txt
|	Pipe. Geeft de output van het ene commando door aan het andere commando. Bijvoorbeeld: `ls | more`
>	Plaatst de output van een commando in een nieuw bestand. Als het bestand al bestaat wordt het overschreven. Bijvoorbeeld: `ls > myfiles.txt`
>>	Voegt de output van een commando achteraan een bestaand bestand toe. Bijvoorbeeld: `echo "Tom 09 123 45 67" >> telefoonnummers.txt`
<	Gebruikt een bestand als input voor een programma. Bijvoorbeeld: `more < telefoon.txt`
;	Scheidingsteken tussen 2 commando's. Bijvoorbeeld: `cd .. ; ls`

Het bestandssysteem in Linux

- Het bestandssysteem in Linux is een boomstructuur die bestaat uit bestanden en directories. De basis van het bestandssysteem is de "/" – directory: de root (verwar dit niet met de root – user). Alle andere directories, bestanden, drives en devices worden vastgemaakt aan deze root.

`/bin`	Essentiële command binaries (programma's) die door alle gebruikers gebruikt worden, worden hier opgeslagen: `bash`, `ls`, `mount`, …
`/boot`	Alle files die nodig zijn om de computer te booten. Deze bestanden veranderen zelden.
`/etc`	Machine specifieke configuratiebestanden. Vroeger stond `/etc` voor etcetera, tegenwoordig staat het voor Editable Text Configuration
`/home`	Locatie van de home directories van de gebruikers
`/lib`	Essentiële shared libraries en kernel modules
`/dev`	Device files. In Linux worden hardware devices gezien als gewone bestanden.

Commando's om te navigeren

`pwd`	Print Working Directory. Toont de huidige directory.
`cd`	Change Directory. Zonder argument, keer je terug naar je home directory
`cd dir`	Ga naar de opgegeven directory. Bijvoorbeeld: `cd /usr/src/linux`
`cd ~`	Keer terug naar de home directory
`cd ..`	Ga naar de parent directory
`cd -`	Ga naar de vorige directory
`ls`	Geef alle bestanden van de huidige directory, in kolomformaat
`ls dir`	Geef alle bestanden van de opgegeven directory
`ls -l`	Geef alle bestanden van de huidige directory, in lang formaat: Elke regel bevat 1 lijn en toont extra informatie over het bestand, zoals datum, grootte, …
`ls -a`	Geef alle bestanden, ook de verborgen bestanden. Deze bestanden beginnen met een . (bijvoorbeeld .bash_history)
`ls -ld dir`	In plaats van de inhoud van de directory, wordt gedetailleerde informatie van de directory getoond.

Werken met bestanden en directories

`file`	Geeft het type bestand weer, of het een ASCII tekstbestand is, of een Linux executable file of …
`cat`	Geeft de inhoud van een bestand op het scherm
`touch`	Eenvoudige manier om een bestand te maken. Bijvoorbeeld: `touch file1`
`head`	Geeft de eerste 10 lijnen van een bestand
`tail`	Geeft de laatste 10 lijnen van een bestand
`head -n`	Geeft de eerste n lijnen van een bestand
`tail -n`	Geeft de laatste n lijnen van een bestand
`cp`	Kopieert een bestand van 1 locatie naar een andere locatie Bijvoorbeeld: `cp winter.txt /tmp`
`mv`	Verplaatst een bestand naar een andere locatie of hernoemt een bestand Bijvoorbeeld: `mv winter.txt herfst.txt`
`rm`	Verwijdert een bestand Bijvoorbeeld: `rm herfst.txt`
`mkdir`	Maakt een directory
`rmdir`	Verwijdert een directory

Bestanden zoeken

`which`	Toont het volledige pad van een shell commando dat zich in je PATH bevindt. Bijvoorbeeld: `which ls` geeft `/usr/bin/ls`
`whereis`	Localiseert het programma, de broncode en de manual page voor een commando. Bijvoorbeeld: `whereis ls` geeft `ls: /bin/ls /usr/share/man/man1/ls.1.gz`
`find`	Een zeer krachtig commando dat gebruikt wordt om bijvoorbeeld te zoeken naar bestanden die voldoen aan een bepaald patroon. Bijvoorbeeld: `find . -name \*mp3` zoekt alle bestanden in de huidige directory en in de subdirectories waarvan de naam eindigt op mp3

Sneltoetsen

`Pijltje op en neer`	Door de meest recente commando's scrollen.
`history`	Dit commando toont de volledige geschiedenis van ingegeven commando's. Elk commando wordt voorafgegaan door een getal. Met behulp van !getal kan je een bepaald commando opnieuw uitvoeren.
`tab`	Als je een gedeelte van een commando of een bestandsnaam ingeeft dat de shell begrijpt, kan je de rest automatisch laten aanvullen door op de TAB toets te drukken
`recente commando's aanvullen met !`	! aangevuld met enkele letters van een recent commando, wordt aangevuld tot dit volledige commando na druk op de Enter toets
`Ctrl-r`	Druk Ctrl-r en type een gedeelte van een recent commando. Als je Ctrl-r blijft drukken worden alle commando's met dit deel achtereenvolgens weergegeven. Als je het juiste commando gevonden hebt, druk je op Enter.

Gebruikers

- Bekijk Chapter 23 in Linux Fundamentals

`id`	Geeft de user id, primary group id en een lijst van alle groepen waartoe je behoort
`useradd`	Een gebruiker toevoegen
`userdel`	Een gebruiker verwijderen
`usermod`	De eigenschappen van een gebruiker wijzigen
`passwd`	Het paswoord van een gebruiker instellen
`/etc/shadow`	Bestand waarin de paswoorden geëncrypteerd worden bijgehouden
`su`	Substitute user
`sudo`	Stelt gebruikers in staat om programma's uit te voeren met privileges van een andere gebruiker, meestal met rootrechten. Dit kan nuttig zijn om andere gebruikers toe te laten administratieve taken uit te voeren (zonder het root passwoord te moeten geven)

Groepen

- Bekijk Chapter 24 in Linux Fundamentals

`groupadd`	Een groep creëren
`/etc/group`	Bestand waarin de namen van de groepen, de group id's en de members van de groep worden bijgehouden
`groupmod`	Om de naam van de groep te veranderen
`groupdel`	De groep verwijderen
`groups`	Geeft de groepen van een user

Bronnen:
http://www.karlin.mff.cuni.cz/~hron/NMNV532/ShellIntro.pdf
Linux Fundamentals - Paul Cobbaut

Werken met bestanden en directories

`mkdir`	To create a directory `-p: makedir will create parent directories as needed`
`rmdir`	To remove a directory `-p: makedir will recursively remove directories`

`rm`	To remove a file
	`-i: to prevent yourself from accidentally removing a file`
	`-r: remove directories and their contents recursively`

Bestanden

`touch`	A way to create a file
`echo`	Display a line of tekst
`cp`	Copy a file You can copy multiple files into a directory: the last argument must be a directory `-r: to copy complete directories (recursive copying of all files in all subdirectories)`
`mv`	To rename a file or to move a file to another directory

Pathname expansion

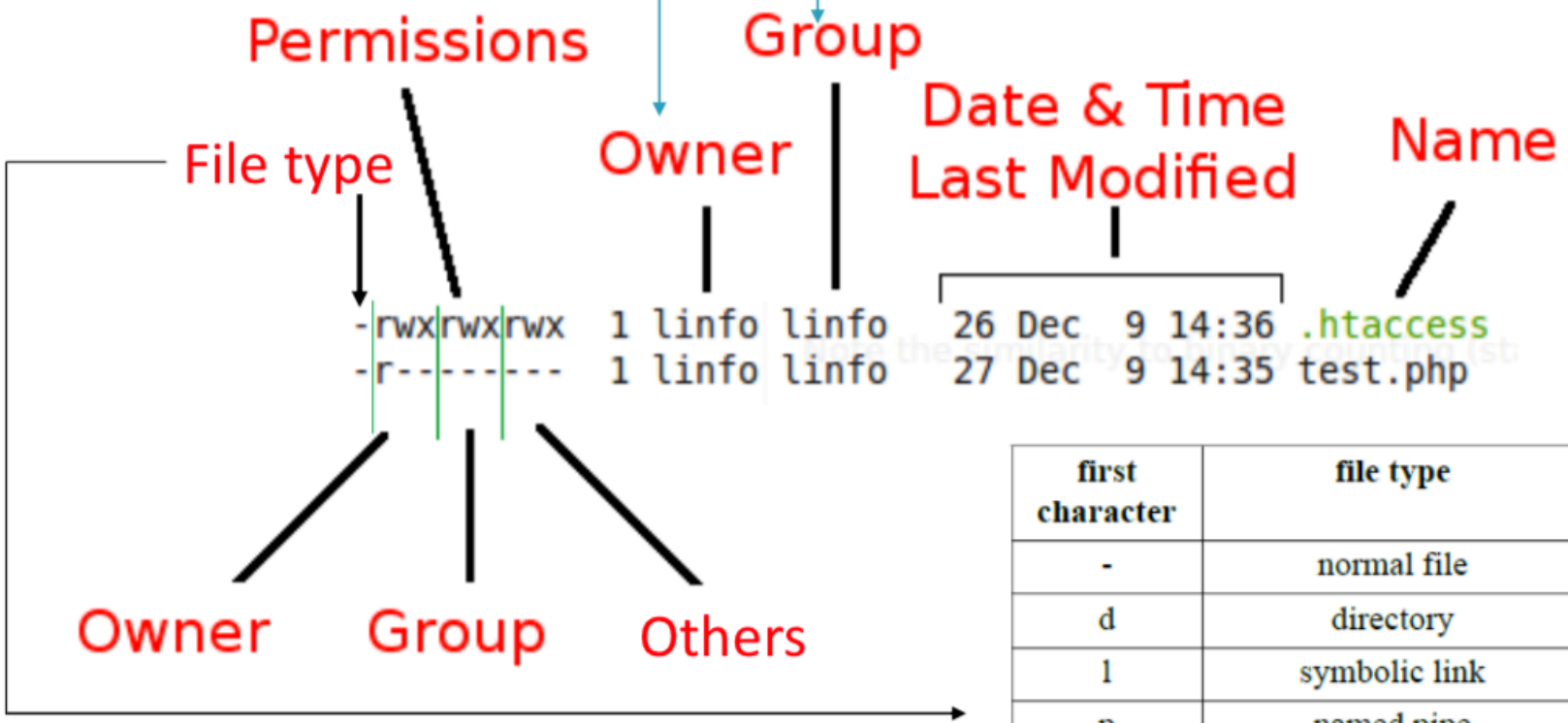
*	Any combination of characters (even none) . For example `ls file*`
?	Exactly one character. For example `ls file?`
[]	Matching any of the characters between []. For example `ls file[0-9]`. For example `ls file[!5]`

Links

`ln`	Create a hard link `-s: create a soft link`

Wie heeft welke rechten?

Elke file en elke directory heeft een owner en een group



first character	file type
-	normal file
d	directory
l	symbolic link
p	named pipe
b	block device
c	character device
s	socket

katholieke hogeschool associatie KU Leuven vives

Wat betekenen die rechten?

permission	on a file	on a directory
r (read)	read file contents (cat)	read directory contents (ls)
w (write)	change file contents (vi)	create files in (touch)
x (execute)	execute the file	enter the directory (cd)

Hoe kan je die rechten instellen?

Permissions can be changed with **chmod**. The first example gives the user owner execute permissions.

chmod wie krijgt wat

chmod u+x

```
paul@laika:~/perms$ ls -l permissions.txt
-rw-r--r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
paul@laika:~/perms$ chmod u+x permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwxr--r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

This example removes the group owners read permission.

chmod g-r

```
paul@laika:~/perms$ chmod g-r permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwx---r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

This example removes the others read permission.

chmod o-r

```
paul@laika:~/perms$ chmod o-r permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwx------ 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

This example gives all of them the write permission.

chmod a+w

```
paul@laika:~/perms$ chmod a+w permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwx-w--w- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

katholieke hogeschool associatie KU Leuven vives

Hoe kan je die rechten instellen?

You don't even have to type the a.

chmod wie krijgt wat

chmod +x

```
paul@laika:~/perms$ chmod +x permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwx-wx-wx 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

You can also set explicit permissions.

chmod u=rw

```
paul@laika:~/perms$ chmod u=rw permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rw--wx-wx 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

Feel free to make any kind of combination.

```
paul@laika:~/perms$ chmod u=rw,g=rw,o=r permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rw-rw-r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

chmod u=rw,g=rw,o=r

katholieke hogeschool associatie KU Leuven vives

Octale notatie (old school notation)

- In plaats van rwx kunnen we de octale notatie gebruiken

Binair	Octaal	Permissie	Representatie
000	0 (0+0+0)	No Permission	---
001	1 (0+0+1)	Execute	--x
010	2 (0+2+0)	Write	-w-
011	3 (0+2+1)	Write + Execute	-wx
100	4 (4+0+0)	Read	r--
101	5 (4+0+1)	Read + Execute	r-x
110	6 (4+2+0)	Read + Write	rw-
111	7 (4+2+1)	Read + Write + Execute	rwx

- Je kan chmod ook met octale notatie gebruiken om de permissies te wijzigen

```
paul@laika:~/perms$ chmod 777 permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwxrwxrwx 1 paul paul 0 2007-02-07 22:34 permissions.txt
paul@laika:~/perms$ chmod 664 permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rw-rw-r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
paul@laika:~/perms$ chmod 750 permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwxr-x--- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

Wat zijn de default permissies?

- De default permissies worden bepaald door umask. Dit specificeert de permissies die je by default níet wil instellen

  Een bestand is by default niet executable. Je moet gebruik maken van chmod +x om een file executable te maken

```
[Harry@RHEL4b ~]$ umask
0002
[Harry@RHEL4b ~]$ touch test
[Harry@RHEL4b ~]$ ls -l test
-rw-rw-r--  1 Harry Harry 0 Jul 24 06:03 test
[Harry@RHEL4b ~]$
```

- Je kan gebruik maken van mkdir –m om de permissies van een directory in te stellen bij het creëren

```
paul@debian5~$ mkdir -m 700 MyDir
paul@debian5~$ mkdir -m 777 Public
paul@debian5~$ ls -dl MyDir/ Public/
drwx------ 2 paul paul 4096 2011-10-16 19:16 MyDir/
drwxrwxrwx 2 paul paul 4096 2011-10-16 19:16 Public/
```

Hoe kan je owner en group wijzigen?

- chgrp: de group van een file wijzigen

```
root@laika:/home/paul# touch FileForPaul
root@laika:/home/paul# ls -l FileForPaul
-rw-r--r-- 1 root root 0 2008-08-06 14:11 FileForPaul
root@laika:/home/paul# chgrp paul FileForPaul
root@laika:/home/paul# ls -l FileForPaul
-rw-r--r-- 1 root paul 0 2008-08-06 14:11 FileForPaul
```

- chown: de owner van een file wijzigen

```
root@laika:/home/paul# ls -l FileForPaul
-rw-r--r-- 1 root paul 0 2008-08-06 14:11 FileForPaul
root@laika:/home/paul# chown paul FileForPaul
root@laika:/home/paul# ls -l FileForPaul
-rw-r--r-- 1 paul paul 0 2008-08-06 14:11 FileForPaul
```

- chown: de owner en de group van een file tegelijk wijzigen

```
root@laika:/home/paul# ls -l FileForPaul
-rw-r--r-- 1 paul paul 0 2008-08-06 14:11 FileForPaul
root@laika:/home/paul# chown root:project42 FileForPaul
root@laika:/home/paul# ls -l FileForPaul
-rw-r--r-- 1 root project42 0 2008-08-06 14:11 FileForPaul
```

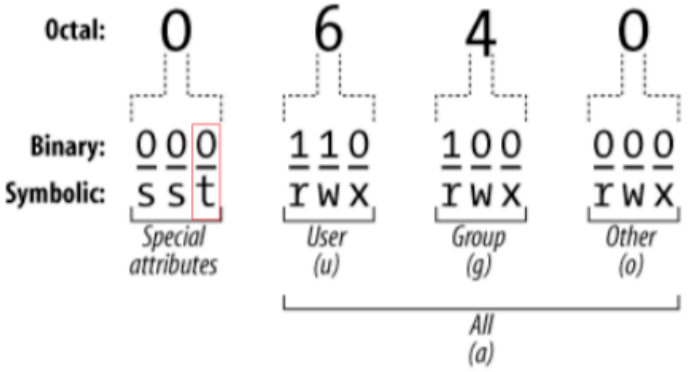
Sticky bit on directory

- Sticky bit on directory: om ervoor te zorgen dat een bestand enkel kan verwijderd worden door de eigenaar
- De sticky bit wordt getoond op dezelfde plaats als de x – permissie voor others.
  - t => x is ook aanwezig
  - T => x is niet aanwezig

```
root@RHELv4u4:~# mkdir /project55
root@RHELv4u4:~# ls -ld /project55
drwxr-xr-x  2 root root 4096 Feb  7 17:38 /project55
root@RHELv4u4:~# chmod +t /project55/
root@RHELv4u4:~# ls -ld /project55
drwxr-xr-t  2 root root 4096 Feb  7 17:38 /project55
root@RHELv4u4:~#
```

- De sticky bit kan ook ingesteld worden met octale permissies

```
root@RHELv4u4:~# chmod 1775 /project55/
root@RHELv4u4:~# ls -ld /project55
drwxrwxr-t  2 root root 4096 Feb  7 17:38 /project55
root@RHELv4u4:~#
```



setgid bit on directory

- setgid: zorgt ervoor dat de group owner van alle files in de directory = de group owner van de directory
- De setgid bit wordt getoond op dezelfde plaats als de x – permissie voor group.
  - s => x is ook aanwezig
  - S => x is niet aanwezig

```
root@RHELv4u4:~# groupadd proj55
root@RHELv4u4:~# chown root:proj55 /project55/
root@RHELv4u4:~# chmod 2775 /project55/
root@RHELv4u4:~# touch /project55/fromroot.txt
root@RHELv4u4:~# ls -ld /project55/
drwxrwsr-x  2 root proj55 4096 Feb  7 17:45 /project55/
root@RHELv4u4:~# ls -l /project55/
total 4
-rw-r--r--  1 root proj55 0 Feb  7 17:45 fromroot.txt
root@RHELv4u4:~#
```

setuid op regular files

- Het instellen van setuid zorgt ervoor dat een executable file wordt uitgevoerd met de permissies van de eigenaar van het bestand in plaats van met de permissies van degene die het bestand uitvoert
- Bijvoorbeeld als setuid is ingesteld voor een executable file van de root, dan kan een gewone gebruiker het bestand ook uitvoeren (als was hij root)
- Bijvoorbeeld
  - Paswoorden worden opgeslagen in /etc/shadow. Dit bestand kan enkel gelezen worden door de root

```
root@RHELv4u4:~# ls -l /etc/shadow
-r--------  1 root root 1260 Jan 21 07:49 /etc/shadow
```

  - Als een gewone gebruiker zijn paswoord aanpast, moet dit bestand aangepast worden. Hoe kan een non-root gebruiker dit doen? Je gebruikt de credentials van de root

```
root@RHELv4u4:~# ls -l /usr/bin/passwd
-r-s--x--x  1 root root 21200 Jun 17  2005 /usr/bin/passwd
```

12

katholieke hogeschool associatie KU Leuven vives

setgid op regular files

- Het instellen van de setgid bit op een executable file zorgt ervoor dat de executable file wordt uitgevoerd met de credentials van de group owner

Werken met variabelen en parameters in een script. (2)

Werken met invoer:

Met het commando **read** kan je tekst inlezen van het toetsenbord en deze tekst opslaan in een variabele.

Voorbeeld:

```
#!/bin/bash
# lees
clear
echo "Geef een naam: ";read naam1
echo
echo "Geef nog een naam: ";read naam2
echo
echo "De eerste naam is $naam1."
echo "De tweede naam is $naam2."
echo
```

```
Geef een naam:
Jan

Geef nog een naam:
Johan

De eerste naam is Jan.
De tweede naam is Johan.
```

katholieke hogeschool associatie KU Leuven vives

Werken met variabelen en parameters in een script. (4)

```
#!/bin/bash
# parameters [parameter1, parameter2, ... parametern]
# Dit script wordt gebruikt om te tonen
# welke parameters zijn gebruikt
clear
echo "De opdrachtnaam is " $0
echo "De eerste parameter is " $1
echo "De negende parameter is " $9
echo
```

```
nathalie@nathalie-laptop:~$ bash parameters.sh Hallo, wie ben jij?
```

```
De opdrachtnaam is  parameters.sh
De eerste parameter is  Hallo,
De negende parameter is
```

katholieke hogeschool associatie KU Leuven vives