Chapter 5 – Manpages

**man command**

Type man and then a command for what you need help with. Press **q**  to quite the man page

**man $configfile**

Most configuration files have their own manual

Then enter the command as follows: example: resolver configuration file:

***man resolv.config***

**man $daemon or man <root-binaries>**

*daemons are background programs.* for example: system.networkd.service

***man system-networkd***

**man -k (apropos)**

shows a list of man pages containing a string

**whatis**

to see the description of a manual page, use ***whatis*** followed by a string

**whereis**

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

For example: ***whereis passwd***

The file would be directly readable by man:

man /usr/share/man/man5/passwd.5.gz

**man sections**

The numbers in the round brackets are section numbers, for example:

### route (8)

Overview of the list:

### 1 Executable programs or shell commands 2 System calls (functions provided by the kernel) 3 Library calls (functions within program libraries) 4 Special files (usually found in /dev) 5 File formats and conventions eg /etc/passwd 6 Games 7 Miscellaneous (including macro packages and conventions), e.g. man(7) 8 System administration commands (usually only for root) 9 Kernel routines [Non standard]

**man $section $file**

Therefore, when you refer to the man page of the passwd comman, it will be written as ***passwd(1)****.* When referring to the passwd file, it will be written as ***passwd(5).*** How to open the man page of the correct section:

### gert@ubuntu:~$ man passwd      # opens the first manual found, here man 1 passwd gert@ubuntu:~$ man 5 passwd    # opens a page from section 5

**man man**

opens man page for man

**mandb**

Should you be convinced that a manpage exists, then you can run ***mandb***on Debian/Mint:

### gert@ubuntu:~$ sudo mandb

**Searching in the man pages**

You can search through a man page by typing / followed by a string (word, letter)

### gert@ubuntu:~$ man ls /free   and push <enter>          (all lowercase) /Free   and push <enter>         (at least one uppercase letter) (the search is case sensitive and finds ‘Free’, not ‘free’.

**How to move between found occurrences**

type ***‘n’***  to go to the next occurrence

type **‘N’** to go to the previous occurrence

**Searching for a whole word (not a string)**

### /\btext\b   and push <enter>

This searches for cases insensitive for the word ‘text’. Example: words like context

**Extra shortcuts**

Type **‘g’** to go to the first line: (can also use arrowkeys, space, or pageup, pagedown)

Type **‘G’** to go to the last line, and type ‘**h’** to get the help of ***man***

Chapter 6 – Working with directories

This chapter is an overview of the most common commands used in linux: **pwd, cd, ls, mkdir, rmdir**.

**pwd**

stands for *print working directory.* This tool displays your current working directory.

**cd**

You can change your current directory by typing ***cd.***

### gert@ubuntu:~$ cd /etc gert@ubuntu:/etc$ pwd /etc gert@ubuntu:/etc$ cd /bin gert@ubuntu:/bin$ pwd /bin

**cd~**

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

**cd ..**

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

### gert@ubuntu:/usr/share/games$ pwd /usr/share/games gert@ubuntu:/usr/share/games$ cd ..

To stay in the current directory, type ***cd .***  (with one dot)

**cd –**

to go back to the previous directory

**absolute and relative paths**

When you type a path starting with a /, then the root of the file tree is assumed. If you don’t type /, then the current directory is the assumed starting point.

Example: being inside ***home/gert***. ***cd home*** = No such file directory, meanwhile ***cd /home*** is correct.

When inside **/home**,

you have to type **cd gert** instead of **cd /gert**

to enter the subdirectory **gert** of the current directory **/home**.

**path completion\_\_\_\_**

The tab key can help you type a path without errors. Typing ***cd /et*** followed by the ***tab*** key, will give you ***cd /etc/*** note: it is case sensitive, so typing /Et will result in nothing**. It is a faster way to type!**

**ls**

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

Frequently used options for ***ls***

***-a*** show all files, including hidden files. When a file starts with a dot, then it is hidden.

***-l*** will give you a long listing

***- h*** It shows the numbers (file sizes) in a more readable format: ***ls -l -h***

**tree**

To get an overview of your directories and files in the form of a tree. You have to install this first with:

***sudo apt install tree  
tree***

**mkdir**

make your own directory. You have to give at least one parameter to ***mkdir***, namely the name of the new directory.

### gert@ubuntu:~$ mkdir mydir gert@ubuntu:~$ cd mydir gert@ubuntu:~/mydir$ ls -al total 8 gert@ubuntu:~/mydir$ mkdir stuff gert@ubuntu:~/mydir$ mkdir otherstuff gert@ubuntu:~/mydir$ ls -al total 8

**mkdir -p**

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

**rmdir**

When a directory is empty, you can remove it with ***rmdir***.

**rmdir -p**

You can also use rmdir to recursively remove directories if they don’t contain any files

Chapter 7 – Working with files

In this chapter we learn how to recognize, create, remove, copy, and move files using commands like

**file, touch, rm, cp, mv, rename**

**all files are case sensitive**

This means ***FILE1*** is different than ***file1***

**file**

A directory is a special kind of file, but it is still a file. The ***file*** utility determines the file type. No extensions used, command line doesn’t care if a file ends in .txt or .pdf. example: ***file pic33.png***

The file command uses a magic file that contains patterns to recognize file types. Magic file is located in **/usr/share/file/magic**. Type **man 5 magic** for more information. It is interesting to point out ***file -s***  foe special files like those in /dev and /proc

**touch**

One way to create an empty file is with ***touch***.

**touch -t**

The ***touch -t*** command can set some properties while creating empty files. Check ***man -t***  for more info.

**rm**

rm is remove forever. Use rm to remove files. When you use rm to remove a file, it is gone forever.

**rm -i**

To prevent yourself from removing a file, use ***rm -i***.

### gert@ubuntu:~$ ls biology file33  file42 gert@ubuntu:~$ rm -i file33 rm: remove regular empty file `file33'? yes

**rm .\***

Will remove hidden files. Do not try these commands in your home folder. It will remove necessary files. By default, ***rm \****  will remove files that do not contain a dot, and are mnot hidden. Two options to delete hidden files:

### Option 1: gert@ubuntu:~$ mkdir demo gert@ubuntu:~$ cd demo gert@ubuntu:~/demo$ touch .hiddenfile1 .hiddenfile2 visiblefile1  visiblefile2 gert@ubuntu:~/demo$ ls -A .hiddenfile1 .hiddenfile2 visiblefile1  visiblefile2 gert@ubuntu:~/demo$ rm \* gert@ubuntu:~/demo$ ls -A .hiddenfile1 .hiddenfile2 gert@ubuntu:~/demo$ rm .\* rm: cannot remove '.': Is a directory rm: cannot remove '..': Is a directory gert@ubuntu:~/demo$ ls -A gert@ubuntu:~/demo$ cd gert@ubuntu:~$ rmdir demo

### Option 2: gert@ubuntu:~$ mkdir demo gert@ubuntu:~$ cd demo gert@ubuntu:~/demo$ touch .hiddenfile1 .hiddenfile2 visiblefile1  visiblefile2 gert@ubuntu:~/demo$ ls -A .hiddenfile1 .hiddenfile2 visiblefile1  visiblefile2 gert@ubuntu:~/demo$ shopt -s dotglob gert@ubuntu:~/demo$ rm \* gert@ubuntu:~/demo$ ls -A gert@ubuntu:~/demo$ shopt -u dotglob gert@ubuntu:~/demo$ cd gert@ubuntu:~$ rmdir demo

The **shopt** is a shell builtin command to set and unset (remove) various Bash shell options.

**rm -r\_\_\_\_**

By default, ***rm-r***  will not remove any non-empty directories. However, ***rm*** accepts several options that allow you to remove any directory. ***rm -rf***  will allow you to remove anything. R means recursive, f means force. If you type ***rm -rf*** while logged on as root, then you will literally erase your entire file system.

**cp**

copy a file.

### gert@ubuntu:~$ ls file42  biology gert@ubuntu:~$ cp file42 file42.copy gert@ubuntu:~$ ls file42  file42.copy  biology

### gert@ubuntu:~$ ls *copy to another directory:* file42  biology gert@ubuntu:~$ mkdir dir42 gert@ubuntu:~$ cp biology dir42/ gert@ubuntu:~$ ls dir42/

### gert@ubuntu:~$ ls *copy complete directories: cp -r* dir42  file42  file42.copy  biology gert@ubuntu:~$ ls /dir42 biology gert@ubuntu:~$ cp -r dir42/ dir33 gert@ubuntu:~$ ls dir33  dir42  file42  file42.copy  biology gert@ubuntu:~$ ls dir33/ biology

### gert@ubuntu:~$ cp file42 file42.copy biology dir42/ gert@ubuntu:~$ ls dir42/ file42  file42.copy  biology *copy multiple files to directory*

### gert@ubuntu:~$ cp file42 dir42/file42.backup gert@ubuntu:~$ ls dir42/ *rename file while cp to another dir.* file42  file42.backup file42.copy  biology

### gert@ubuntu:~$ cp biology file42 gert@ubuntu:~$ cp biology file42 gert@ubuntu:~$ cp -i biology file42 cp: overwrite `file42'? n gert@ubuntu:~$ *prevent cp from overwriting: cp -i*

**mv**

use mv to rename a file or move it to another directory. When you need to rename only one file, then mv is the preferred command. The same command ***mv*** can be used to rename directories (line2)

### gert@ubuntu:~$ mv file42 file33

### gert@ubuntu:~$ mv dir33 backup

**mv -i**

will ask permission to overwrite an existing file.

**rename**

Only works on some versions of Linux systems. Try to use ***mv*** instead, when renaming or moving only a couple of files. The ***rename*** command on the Debian-family distributions uses regular expressions (regular expression or *shor* *regex* are explained in a later chapter) to rename many files at once.

### gert@ubuntu:~/test42$ ls abc.txt  file33.txt  file42.txt anotherfile.docx gert@ubuntu:~/test42$ rename 's/\.txt/.png/' \*.txt gert@ubuntu:~/test42$ ls abc.png  file33.png  file42.png  anotherfile.docx

Chapter 8 – 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**

**head**

You can use this to display the first 10 lines of a file.

### gert@ubuntu~$ head /etc/passwd

You can also be more specific: (the first n lines)

### gert@ubuntu~$ head -4 /etc/passwd

or the first n bytes:

### gert@ubuntu~$ head -c14 /etc/passwd

**tail**

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

### gert@ubuntu~$ tail /etc/services

### gert@ubuntu~$ tail -3 /etc/services

**cat \_\_\_**

One of the most universal tools. All it does is copy standard input into standard output.

### gert@ubuntu:~$ cat /etc/resolv.conf domain linux-training.be

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

### gert@ubuntu:~$ echo one > part1 gert@ubuntu:~$ echo two > part2 gert@ubuntu:~$ echo three > part3 gert@ubuntu:~$ cat part1 one gert@ubuntu:~$ cat part2 two gert@ubuntu:~$ cat part3 three gert@ubuntu:~$ cat part1 part2 part3 one two three gert@ubuntu:~$ cat part1 part2 part3 > all gert@ubuntu:~$ cat all one two three

You can use ***cat*** to create flat text files. Type ***cat*** *> name, then type a selection of lines, separating them with enter. After the last line, hold ctrl D. The ctrl D combo will send an EoF (End of File) to the running process ending the cat command.*

### gert@ubuntu:~$ cat > winter.txt It is very cold today!                   CTRL+d gert@ubuntu:~$ cat winter.txt It is very cold today! gert@ubuntu:~$

You can choose an end marker for ***cat***  with << This construction is called ***here documents*** and will end the ***cat***  command.

### gert@ubuntu:~$ cat << stop  > hot.txt > It is hot today! > Yes it is summer. > stop gert@ubuntu:~$ cat hot.txt It is hot today! Yes it is summer.

**tac**

It is cat backwards. Cat count = one, two tree, four. Tac count = four, three, two, one.

**more and less**

***more*** is useful for displaying files that take up more than one screen. More will allow you to see the contents of the file page by page. Use ***space*** to see next page, ***q*** to quit. Some people prefer the ***less*** command to more, because you can go up in the document with ***UP*** or ***arrow key***.

**strings**

You can display readable ascii strings found in binary files. The following example located the ***ls*** binary, then displays readable strings in the binary file (output is truncated).

### gert@ubuntu:~$ which ls /bin/ls gert@ubuntu:~$ strings /bin/ls /lib/ld-linux.so.2 librt.so.1 \_\_gmon\_start\_\_ \_Jv\_RegisterClasses clock\_gettime libacl.so.1

Chapter 9 – The Linux file tree

This chapter looks at the most common directories in the Linux file tree.

Filesystem hierarchy standard

Many Linux distributions partially follow the FHS. The FHS

is available online at **http://www.pathname.com/fhs/** where we read: "The filesystem hierarchy standard has been designed to be used by Unix distribution developers, package developers, and system implementers. However, it is primarily intended to be a reference and is not a tutorial on how to manage a Unix filesystem or directory hierarchy.”

**man hier**

This command will help you find out about the directory structure inside your computer.

The root directory

All Linux systems have a directory structure that starts at the root directory. The root directory is represented by ***/***

Contents of the root directory

### gert@ubuntu~$ ls / bin    dev   lib    libx32      mnt   root  snap      sys  var boot   etc   lib32  lost+found  opt   run   srv       tmp cdrom  home  lib64  media       proc  sbin  swapfile  usr

**/boot**

Contains all files needed to boot your computer. ***boot/grub*** is found here, which contains **/boot/grub/grub.cfg** which defines the boot menu that is displayed before the kernel starts.

**/bin**

Binaries are files that contain compiled source code (or machine code). Sometimes binaries are called ***executables***. The /bin contains binaries for use by all users. Ls /bin to view common commands

### gert@ubuntu~$ ls /bin /bin/cal                     /bin/chvt /bin/calendar                /bin/ciptool /bin/calibrate\_ppa           /bin/ckbcomp /bin/canberra-gtk-play       /bin/cksum

**/sbin**

Contains binaries to configure the operating system. Many of the system binaries require root privileges to perform certain tasks. Example: add user, partition a disk, and create a file system

### gert@ubuntu:~$ ls -l /sbin/useradd /sbin/fdisk /sbin/mkfs -rwxr-xr-x 1 root root 153880 jul 21 09:49 /sbin/fdisk -rwxr-xr-x 1 root root  14568 jul 21 09:49 /sbin/mkfs -rwxr-xr-x 1 root root 147160 mei 28 08:37 /sbin/useradd

**/etc**

These are **machine specific configuration files,** stands for ‘editable text configuration’. Many times the name of configuration files are the same as the application, daemon, or protocol with .conf added as an extension:

### gert@ubuntu:~$ ls /etc/\*.conf /etc/adduser.conf          /etc/host.conf        /etc/pnm2ppa.conf /etc/apg.conf              /etc/kernel-img.conf  /etc/popularity-contest.conf

/etc/skel

The skeleton directory is copied to the home directory of a newly created user. It usually contains hidden files like .bashrc script:

### gert@ubuntu:~$ ls -A /etc/skel/ .bash\_logout  .bashrc  .profile

**/etc/sysconfig**

This directory, not mentioned in the FHS, contains a lot of Red Hat Enterprise Linux config files.

### gert@ubuntu:~$ ls /etc/sysconfig anaconda         firewalld         libvirtd         raid-check  smartmontools atd              grub              man-db           rhn         snapd

**/home**

Users can store personal or project data under /home. Directory is usually named after the user.

It also serves as a location to store the user ***profile.*** Typical user profile contains many hidden files. The hidden files of user profiles contain settings specific for that user.

### gert@ubuntu:~$ ls -a /home/gert .              .cache     .gnupg            Pictures  .sudo\_as\_admin\_successful ..             .config    .lesshst          .profile  Templates

**/root**

Root is the default location for personal data and profile of the root user.

**/srv**

Data that is served by your system. The FHS allows locating cvs, rsync, www, and ftp data in this location. Webserver data is often not saved in srv, but in /var/www.

**/lib**

Binaries found in /bin and /sbin often use shared libraries located in /lib.

**/media**

Serves as a mount point for removable media devices such as USB sticks, CD-ROMS, digital cameras, and other usb attached devices.

**/mnt**

Should be empty and only used for temporary mount points. You will likely encounter some systems that have more than one directory created inside /mount to be used for various local and remote file systems.

**/opt**

The purpose is to store optional software. In many cases this is software from outside the distribution repository. You may find an empty /opt directory on many systems. A large package can install all its files in **/bin, /lib, /etc** subdirectories within **/opt/$packagename/**. If for example the package is called **wp**, then it installs in **/opt/wp**, putting binaries in **/opt/wp/bin** and manpages in **/opt/wp/man**.

**/tmp**

It is used to store temporary data when needed. Data stored here either uses disk space or RAM, both of which are managed by the operating system. Never use /tmp to store important data. In some distributions of Linux /tmp is cleaned at every boot.

**/dev**

Device files in /dev are not located on the hard disk. The /dev directory is populated with files as the kernel is recognizing hardware devices.

**/dev/null**

The Black hole. It has unlimited storage, but nothing can be retrieved from it. Anything writte to dev/null will be discarded. Can be useful to discard unwanted outputs from commands.

### gert@ubuntu:~$ echo Hello World > /dev/null gert@ubuntu:~$

**/proc**

Proc is another special directory, appearing to be ordinary files, but not taking up any disk space. It is a view of the kernel, what the kernel manages, and is a means to interact with it directly. When you ask for a file, the kernel will add its data real time to the file.

**/sys**

The sys directory contains kernel information about the system. The /proc filesystem maps mostly to the process table. People found this very useful, and started adding other things to it as well - cpuinfo, memory statistics, device information… but it also raised the issue that /proc was SUPPOSED to be for the process table… So they added another instance, and named it “sys” aimed to hold system information. But being newer, the entries already added to /proc were allowed to remain as moving them would cause issues with other user tools that processed those files.

**/usr**

It stands for Unix System Resources. The /usr hierarchy should contain shareable, read only data.

**/var/www**

On some distros the website data is saved in /var/www instead of /srv.

**/var/log**

The /var/log serves as central point to contain all log files.

**/var/log/syslog\_\_\_\_**

A typical file to check when troubleshooting on Debian derivatives like ubuntu. By default it will contain information on what just happened to the system. It is the file where almost every service will log on to.

### gert@ubuntu:~$ grep dhcp4 /var/log/syslog Oct 18 00:01:46 ubdesk NetworkManager[750]: <info>  [1602972106.1005] dhcp4 (ens33): canceled DHCP transaction Oct 18 00:01:46 ubdesk NetworkManager[750]: <info>  [1602972106.1005] dhcp4 (ens33): state changed extended -> done Oct 18 00:01:46 ubdesk NetworkManager[750]: <info>  [1602972106.1035] dhcp4 (ens33): activation: beginning transaction (timeout in 45 seconds)

**/var/log/messages**

A typical file to check when troubleshooting on Red Hat (an derivatives) is the /var/log/messages file. By default this file will contain information on what just happened to the system. It is the file where almost every service will log on to.

Chapter 10 – Commands and arguments

This chapter introduces shell expansion, by taking a close look at commands and arguments. Knowing shell expansion is important because many commands on Linux are processed most likely are changed by the shell before they get executed. The command line interface/shell on most Linux systems is called bash, which stands for Bourne against shell. The Bash cell incorporates features from **sh**, the original Bourne shell, **csh**, the C shell, and **ksh**, the Korn shell. This chapter frequently uses the echo command to demonstrate shell features.

### gert@ubuntu:~$ echo Burtonville Burtonville

**arguments**

primary feature of shell = command line scan. When you enter command at command prompt, shell will scan input, cut it up into arguments. While scanning, shell may make changes to line = shell expansion. When the shell has finished scanning and modifying the line, it will be executed.

**white space removal**

The first argument is the command to be executed, the following arguments pertain to the command. Any white space is removed. The **echo** command will display each argument it receives from the shell. The **echo** command will also add a new white space between the arguments it received.

**Single quotes**

You can prevent white space removal by using quotes. The contents of the quoted string are considered as one argument.

### gert@ubuntu:~$ echo 'A line with      single    quotes' A line with      single    quotes

**Double quotes**

Achieve the same thing as single quotes. When using variables, single and double quotes make a difference.

**echo and quotes**

quoted lines can include special characters that are recognized by the echo command (for example ***echo -e***), here are examples using ***/n*** and ***/t*** for newline and tab:

### gert@ubuntu:~$ echo -e "A line with \na newline" A line with

### gert@ubuntu:~$ echo -e "A line with \ta tab" A line with     a tab

**commands**

Not all commands are external to the shell, some are built in. External programs are programs that have their own binary, and reside somewhere within the file system. Many external commands are located in /bin or /sbin. Builtin commands are an important part of the shell program itself.

To find out whether a command is builtin, or whether it is an external command. The third type of output you might see is ‘aliased’. This shows whether the command is aliased or not.

### gert@ubuntu:~$ type cd *built in* cd is a shell builtin gert@ubuntu:~$ type cat *external command* cat is /bin/cat gert@ubuntu:~$ type ls *aliased* ls is aliased to `ls --color=auto'

Some commands are both built-in and external. When you run the command, the built-in shell version will always take priority. In order to run the external command, you must enter the path to the command:

### gert@ubuntu:~$ type -a echo echo is a shell builtin echo is /bin/echo gert@ubuntu:~$ /bin/echo Running the external echo command...

**which**

the command ***which*** will search for binaries in the $PATH environment available. In the example below it can be seen that ***cd*** is built in, because it is not shown, but the others are external.

### gert@ubuntu:~$ which cp ls cd mkdir pwd /bin/cp /bin/ls /bin/mkdir /bin/pwd

**alias\_\_\_**

Often used to create an easier to remember name for a command.

### gert@ubuntu:~$ cat count.txt gert@ubuntu:~$ alias dog=tac gert@ubuntu:~$ dog count.txt

**abbreviate commands**

One can use an alias to abbreviate an existing command. For example: ***alias*** ***c = ‘clear’.***

**Default options**

For example, setting an alias for rm -i

### gert@ubuntu:~$ touch winter.txt gert@ubuntu:~$ alias rm='rm -i' gert@ubuntu:~$ rm winter.txt rm: remove regular empty file `winter.txt'? no

**viewing alias**

Providing no other command will give you an overview of the aliases being used.

### gert@ubuntu:~$ alias c ll alias c='clear' alias ll='ls -lh --color=auto'

**unalias,** undoes an alias: ***unalias ‘name’***

Chapter 11 – Control operators

In this chapter we put more than one command on the line using control operators. Also related parameters $? And similar special characters.

**; semicolon**

You can put two commands on the same line by separating them with a semicolon.

### gert@ubuntu:~$ echo Hello ; echo World Hello World

**& ampersand**

When a line is finished with ***&***, the shell will not wait for the command to finish. It will run in the background, and a message will appear when the command is finished.

**$?**

It is the exit code of the previous command. Actually **$?** is a shell parameter and not a variable, since you cannot assign a value to **$?**.

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

**&& double ampersand**

The shell will interpret && as AND. The second command is only executed if the first one succeeds. Another example of the same **logical AND** principle. This example starts with a working **cd** followed by **ls**, then a non-working **cd** which is **not** followed by **ls**.

### gert@ubuntu:~$ echo first && echo second first second

### gert@ubuntu:~$ cd gen && ls file1  file3  File55  fileab  FileAB   fileabc file2  File4  FileA   Fileab  fileab2 gert@ubuntu:~/gen$ cd gen && ls -bash: cd: gen: No such file or directory

**|| double bar**

This means OR. The second command is only executed when the first one fails.

### gert@ubuntu:~$ echo first || echo second ; echo third first third

**combining && and || \_\_\_\_**

You can combine the AND and OR to write an *if-then-else* structure. This command uses echo to display whether the rm command was successful:

### gert@ubuntu:~/test$ rm file1 && echo It worked! || echo It failed! It worked!

**# pound sign**

Everything written after the pound sign will be ignored. It is useful to write a comment.

**\ escaping special characters**

The backslash \ character enables the use of control characters without the shell interpreting it: Aka neglect

### gert@ubuntu:~$ echo hello \; world hello ; world

**end of line backslash**

Lines ending with backslash are continued on the next line

Chapter 12 – Shell variables

In this chapter we learn to manage environment variables in the shell. These variables are often needed by applications

**$ dollar sign**

Another important character interpreted by the shell is the dollar sign **$**. The shell will look for an **environment variable** named like the string following the **dollar sign** and replace it with the value of the variable (or with nothing if the variable does not exist).

### gert@ubuntu:~$ echo This is the $SHELL shell This is the /bin/bash shell gert@ubuntu:~$ echo This is $SHELL on computer $HOSTNAME This is /bin/bash on computer ubuntu

* The shell variables are case sensitive

**Creating variables**

This example creates the variable **MyVar** and sets its value. It then uses **echo** to verify the value. Be aware that it is not allowed to type spaces around the equal sign!

### gert@ubuntu:~$ MyVar=555 gert@ubuntu:~$ echo $MyVar 555

**quotes**

double quotes allow the process above to take place, single quotes doesn’t.

### gert@ubuntu:~$ city=Burtonville gert@ubuntu:~$ echo "We are in $city today." We are in Burtonville today. gert@ubuntu:~$ echo 'We are in $city today.' We are in $city today.

**set**

You can use the **set** command to display a list of environment variables. On Ubuntu and Debian systems, the **set** command will also list shell functions after the shell variables. Use **set | less** to see the variables then.

**unset**

use unset to remove a variable from your shell environment

**$PS1\_\_\_\_**

This variable determines your shell prompt. The ***bash manual*** has a reference on which extensions work. ***/u***  for username, ***/w*** for the working directory. Look in ***man bash*** for more references.

### gert@ubuntu:~$ PS1=prompt prompt

To void unrecoverable mistakes you can set user prompts a different colour than the root prompt. The way to do that is: Add the following to .bashrc

### RED='\[\033[01;31m\]' WHITE='\[\033[01;00m\]' GREEN='\[\033[01;32m\]' BLUE='\[\033[01;34m\]' PS1="$GREEN\u@\h$WHITE:$BLUE\w$WHITE\$ "

**$PATH**

The path variable determines where the shell is looking for commands. It contains a list of directories. If you want the shell to look in the current directory, add a . : ***$PATH: .***

### gert@ubuntu:~$ echo $PATH /usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:

Your path might be different when using ***su***  instead of ***su -*** . The reason is that su – will take on the environment of the target user.

**env**

The ***env*** command without options will display a list of exported variables. The difference with ***set*** is that ***set*** lists all the variables, including those not exported to child shells. But ***env*** can also be used to start a clean shell (a shell without any inherited environment). The ***env -i*** command clears the environment for the subshell.

Notice in this screenshot that ***bash*** will set the ***$SHELL*** variable on startup.

### gert@ubuntu:~$ bash -c 'echo $SHELL $HOME $USER' /bin/bash /home/gert gert gert@ubuntu:~$ env -i bash -c 'echo $SHELL $HOME $USER' /bin/bash

You can use the ***env*** command to set ***$LANG***, or any other variable, for one instance of bash.

**export**

You can export shell variables to other child shells with this command, but it will not export to the parent shell.

**delineate variables**

Sometimes curly brackets are needed to resolve syntax issues with the dollar sign:

### gert@ubuntu:~$ prefix=Super gert@ubuntu:~$ echo Hello $prefixman and $prefixgirl Hello  and gert@ubuntu:~$ echo Hello ${prefix}man and ${prefix}girl Hello Superman and Supergirl

**unbound variables**

when a variable is unbound, the output will display nothing. In the bash shell ***set -u*** is identical to ***set -o nounset***and likewise***set +u***is identical to ***set +o nounset***. These commands help display an error message to let you know that a variable is empty.

Chapter 13 – Shell embedding and options

**Shell embedding**

The command line can spawn new processes containing a fork of the current shell. You can use variables to prove that shells are created.

### gert@ubuntu:~$ date do okt 12 14:51:52 CEST 2017 gert@ubuntu:~$ date +%A donderdag gert@ubuntu:~$ echo Het is vandaag $(date +%A) Het is vandaag donderdag

**backticks**

You can use **`** instead of dollar bracket to embed: **`date +%A`**

**backticks or single quotes**

Don’t confuse backticks with single quotes. On QWERTY: backticks are left of 1, single quotes left of enter.

Chapter 14 – Shell History

**Repeating the last command**

To repeat last command in bash, type **!!** (pronounced  **bang bang**)

**repeating other commands**

You can repeat other commands using one bang followed by one or more chars. Shell will repeat last command starting with those chars:

### gert@ubuntu:~$ touch file42 gert@ubuntu:~$ cat file42 gert@ubuntu:~$ !to touch file42

**history**

Use history to display shell command history, or ***history n*** to see last n commands (e.g. ***history 13*** *(for last 13 cmds))*

**!n**

When typing this the shell will echo the command and execute it:

### gert@ubuntu:~/test$ !43 ls file1  file2  summer.txt  winter.txt

**ctrl -r**

Can use this to search in the history. It finds the last command containing the chars that will have followed ctrl r in the shell. Example: ctrl – r apt, might result in sudo apt install tree, if that was the last command used.

**$HISTSIZE**

Determines the number of commands that will be remembered in your current environment. Default is 500/1000. You can change it:

### gert@ubuntu:~$ HISTSIZE=15000 gert@ubuntu:~$ echo $HISTSIZE 15000

**$HISTFILE**

Points to the file that contains your history. The bash shell defaults this value to **~/.bash\_history**.

**$HISTFILESIZE**

Number of commands kept in your history file can be set using this command.

**prevent recording a command**

You can prevent a command from being recorded in history using a space prefix:

### gert@ubuntu:~$ \_echo def def gert@ubuntu:~$ echo ghi ghi gert@ubuntu:~$ history 2 9502  echo ghi 9503  history 2

**regular expressions**

It is possible to use regular expressions when using **!** to repeat commands. Following command switches 1 into 2:

### gert@ubuntu:~/test$ cat file1 gert@ubuntu:~/test$ !c:s/1/2/ cat file2 hello

Chapter 15 – File Globbing

**\*asterisk**

Is interpreted as a sign to generate filenames, matching the asterisk to any combination of chars. When no path is given, shell will use filenames in current dir. See ***man glob(7)***for more info. By default hidden files are not included in **\***. If shell option **dotglob** is on, then hidden files will also be seen.

### gert@ubuntu:~/glob$ ls file1  file3  file55  FileA   Fileab  fileabc file2  File4  File55  fileab  FileAB gert@ubuntu:~/glob$ ls \*ile55 file55  File55 gert@ubuntu:~/glob$ shopt -s dotglob gert@ubuntu:~/glob$ shopt dotglob dotglob          on gert@ubuntu:~/glob$ ls \* .file1 file1  file3  file55  FileA   Fileab  fileabc .file2 file2  File4  File55  fileab  FileAB  .fileabc

**? question mark**

Is interpreted by shell as a sign to generate filenames matching the question mark with exactly one char.

**gert@ubuntu:~/glob$** ls  
file1  file2  file3  File4  Filo4  File55  FileA  fileab  Fileab  FileAB  fileabc  
**gert@ubuntu:~/glob$** ls Fil?4  
File4  Filo4

**[] square brackets**

Is interpreted as a sign to generate filenames matching any chars between [ and ]. Order is not important. Each bracket is replaces by exactly one char. You can also exclude chars from a list between [] with **!**. Combinations are allowed.

### gert@ubuntu:~/glob$ ls File[A5] File5 FileA gert@ubuntu:~/glob$ ls File[A5][5b] File55 gert@ubuntu:~/glob$ ls file[a5][5b][abcdefghijklm] fileabc gert@ubuntu:~/glob$ ls file1  file2  file3  File4 file55 File55  FileA  fileab  Fileab  FileAB  fileaZ  fileabc gert@ubuntu:~/glob$ ls file[a5][!Z] fileab gert@ubuntu:~/glob$ ls file[!5]\* file1  file2  file3  fileab  fileaZ  fileabc gert@ubuntu:~/glob$ ls file[!5]? fileab  fileaZ gert@ubuntu:~/glob$

**a-z and 0-9 ranges**

bash shell will also understand ranges in between brackets. Example: **ls file[0-9]: file1 file 2 file 3**

**shell option globasciiranges** **and []**

default is set to on, which means use of default will provide case sensitivity while globbing. Otherwise if ***globasciiranges*** is off case sensitivity will be ignored.

**$LANG and []**

Even is ***globasciiranges*** is off, it does not guarantee that range will be case sensitive. Some languages include lower case letters in an upper case range. If you want to be sure that both upper and lowercase letters will be included, you can combine both in range: ***ls File[a-zA-Z]***

Chapter 16 – I/O redirection

**stdin stdout stderr**

Bash shell has three basic streams, input output and error. Keyboard serves as stdin, and stdout/stderr both go to display.

**Output redirection**

**> stdout**

The shell will see the > and clear the file. Since this happends before resolving argument 0, the file will have been erased even if the command line fails:

### gert@ubuntu:~$ cat winter.txt It is winter! gert@ubuntu:~$ zcho It is cold today! > winter.txt -bash: zcho: command not found gert@ubuntu:~$ cat winter.txt gert@ubuntu:~$

**>> append**

Use this to append output to a file:

### gert@ubuntu:~$ echo It is cold today! > winter.txt gert@ubuntu:~$ echo It is winter! >> winter.txt gert@ubuntu:~$ cat winter.txt It is cold today! It is winter!

**Error redirection**

**2>sdterr**

Command line below redirects **stdout** to a file, and **stderr** to **dev/null**.

### gert@ubuntu:~$ find / > allfiles.txt  2> /dev/null

**2>&1**

This redirects both ***sdtout*** and ***stderr*** to the same file. The order of directions is significant. Second command directs stdout and stderr to file dirlist, while 3rd command directs only stdout to file dirlist, because stderr was redirected to wherever stdout was redirected before stdout was redirected to dirlist:

### gert@ubuntu:~$ find / > allfiles\_and\_errors.txt 2>&1 ls > dirlist 2>&1 ls 2>&1 > dirlist

**output redirection and pipes\_\_**

By default you can’t grep inside ***stderr*** while using pipes on cmd line, because only ***stdout*** is passed. With ***2>&1*** you can force ***stderr*** to go to ***stdout***. You cannot use both ***1>&2*** and ***2>&1*** to switch ***stdout*** and ***stderr***. You need a third stream to switch ***stdout*** and ***stderr*** if you want to use filters on the error stream:

### gert@ubuntu:~$ rm file42 file33 file1201 | grep file42 rm: cannot remove ‘file42’: No such file or directory gert@ubuntu:~$ rm file42 file33 file1201 2>&1 | grep file42 rm: cannot remove ‘file42’: No such file or directory gert@ubuntu:~$ rm file42 file33 file1201 2>&1 1>&2 | grep file42 rm: cannot remove ‘file42’: No such file or directory gert@ubuntu:~$ echo file42 2>&1 1>&2 | sed 's/file42/FILE42/' FILE42 gert@ubuntu:~$ rm file42 | sed 's/file42/FILE42/'     rm: cannot remove ‘file42’: No such file or directory gert@ubuntu:~$ rm file42 3>&1 1>&2 2>&3 | sed 's/file42/FILE42/ rm: cannot remove ‘FILE42’: No such file or directory

**joining stdout and sterr**

The ***&>*** construction will redirect both ***stdout*** and ***stderr*** in one stream to a file. The **|&** construction will put both ***stdout*** and ***stderr*** in one stream and give it to the next filter via the pipe.

**input redirection**

**< stdin**

This is used to redirect ***stdin***.

### gert@ubuntu:~$ cat < text.txt one two gert@ubuntu:~$ tr 'onetw' 'ONEZZ' < text.txt ONE ZZO

**<< here document**

Sometimes called “here is document” is a way to append input until a certain sequence is encountered.

### gert@ubuntu:~$ cat <<einde > text.txt > brel > einde gert@ubuntu:~$ cat text.txt brel

**<<< here string**

Can be used to directly pass strings to a command The result is same as using **echo string | command**, but with one less process running:

### gert@ubuntu:~$ base64 <<< linux-training.be bGludXgtdHJhaW5pbmcuYmUK gert@ubuntu:~$ base64 -d <<< bGludXgtdHJhaW5pbmcuYmUK linux-training.be

**quick file clear**

The quickest way to clear a file:

### >winter.txt

**Chapter 17 – Filters**

Commands that are created to be used with a pipe are often called filters. Can be used as building blocks.

**cat |**

When between pipes, cat does nothing, except put ***stdin*** on ***stdout***.

**tee**

The ***tee*** filter puts ***stdin*** on ***stdout*** and also into a file. ***tee*** is the same as ***cat***, except that it has two identical outputs.

### gert@ubuntu:~/pipes$ tac count.txt | tee temp.txt | tac one two three four five

**grep**

To filter lines of text containing(or not) a certain string.

### gert@ubuntu:~/pipes$ cat tennis.txt Justine Henin, Bel Venus Williams, USA gert@ubuntu:~/pipes$ cat tennis.txt | grep Williams Venus Williams, USA gert@ubuntu:~/pipes$ grep Williams tennis.txt //*without using cat* Serena Williams, usa

**grep -i**

filters in a case insensitive way

**grep -v**

outputs lines not matching string

**grep -vi**

outputs lines not matching string in case sensitive way

**grep -A1**

One line after the result is displayed

**grep -B1**

displays one line before result

**grep -C1**

one line before and one after matching string are displayed

### gert@ubuntu:~/pipes$ grep -C1 Henin tennis.txt Kim Clijsters, BEL Justine Henin, Bel Serena Williams, usa

**cut**

can select coloumns from files, depending on delimiter or count of bytes. When using space as delimiter you have to ***“ “*** the space.

### gert@ubuntu:~/pipes$ cut -d" " -f1 tennis.txt Amelie Kim Justine Serena Venus

**tr\_**

Can translate characters. Shows all occurrences of e to E (example):

### gert@ubuntu:~/pipes$ cat tennis.txt | tr 'e' 'E' AmEliE MaurEsmo, Fra Kim ClijstErs, BEL

can set letters to uppercase by defining ranges: **cat tennis.txt | tr ‘a-z’ ‘A-Z’**

### AMELIE MAURESMO, FRA

Convert newlines to horizontal spaces: **cat count.txt | tr ‘\n’ ‘ ’**

### one two three four five

**tr -s \_\_**

Is used to *squeeze* multiple occurrences of a character to one: **cat spaces.txt | tr -s ‘ ‘**

### one    two        three --> one two three     four               five     six four five six

**tr rot 13**

“Old Cesaer way” of encrypting with exchanging all letters to letter+13 in alphabet:

**cat count.txt | tr ‘a-z’ ‘n-za-m’** **> encrypted.txt**

**cat encrypted .txt** (content of file: one two three four five):

*bar gjb guerr sbhe svir*

**wc**

counting words, lines, and characters: ***(-w, -l, -c)***

**sort**

will default to alphabetical sort

**sort -k1** or **sort -n**

Will sort coloumn one from content of file. Difference between alphabetical and numerical sort: numerical sort sorts the numbers in ascending order.

### gert@ubuntu:~/pipes$ sort -n -k3 country.txt Belgium, Brussels, 10 Italy, Rome, 50 France, Paris, 60

**uniq**

with uniq you can remove duplicates from a sorted list: ***sort music.txt* | *uniq***

**uniq -c**

It is also possible to count occurrences:

### gert@ubuntu:~/pipes$ sort music.txt |uniq -c      1 Abba      1 Brel      2 Queen

**comm**

used to compare streams or files. Default output is three coloumns: Example: Abba, Cure, and Queen saved in both lists, Bowie and Sweet are only in first list, and Turner only in second:

### gert@ubuntu:~/pipes$ comm list1.txt list2.txt                Abba Bowie                Cure                Queen Sweet        Turner

output of comm can be controlled with **­–[1,2,3]**, The digits point out which output coloumns should not be displayed.

### gert@ubuntu:~/pipes$ comm -23 list1.txt list2.txt Bowie Sweet

**sed**

**s**tream **ed**itor can perform editing functions using regular expressions. Add **g** for global replacements, add **d** to remove lines from stream containing a character:

### gert@ubuntu:~/pipes$ echo level5 | sed 's/5/42/' level42 gert@ubuntu:~/pipes$ echo level5 level7 | sed 's/level/jump/g' jump5 jump7 gert@ubuntu:~/test42$ cat tennis.txt | sed '/BE/d' Venus Williams, USA Martina Hingis, SUI

**pipe examples**

**who | wc**

how many users are logged on?:

### gert@ubuntu:~/pipes$ who root     tty1         Jul 25 10:50 gert     pts/0        Jul 25 09:29 (laika) harry    pts/1        Jul 25 12:26 (barry) gert     pts/2        Jul 25 12:26 (pasha) gert@ubuntu:~/pipes$ who | wc -l 4

**who | cut -d ‘ ‘ -f1 | sort | uniq**

sorted list of logged on users, but every user only once:

### gert@ubuntu:~/pipes$ who | cut -d' ' -f1 | sort | uniq gert harry root

**grep | cut**

Display a list of all bash user accounts in detail:

### gert@ubuntu:~$ grep bash /etc/passwd | cut -d: -f1 root

**Chapter 18 – Basic Unix tools**

Commands to find, locate and compress files.

**find**

example: find all files in /etc and put the list in etcfile.txt 🡪 ***find /etc > etcfiles.txt***

example: find all files and put the list in allfilex.txt 🡪 ***find / > allfiles.txt***

example: find files that end in .conf in current dir 🡪 ***find . name “\*.conf”***

example: find files of type file that end in .conf 🡪 ***find . -type f -name “\*.conf”***

example: find files that are newer than file42.txt 🡪 ***find . -newer file42.txt***

example: Look for \*.odf files and copy them to /backup/

***find /data -name “\*.odf” -exec cp {} /backup/ \;***

example: remove \*.odf files if you approve of it for every file found:  
***find /data -name “\*.odf” -ok rm {} \;***

**locate**

Uses the database to find files, so if you haven’t updated the database, locate will not find files in question.

### gert@ubuntu:~$ ls all\* allfiles.txt gert@ubuntu:~$ ls backups/all\* allfiles.txt.backup gert@ubuntu:~$ sudo updatedb         #om de database up-to-date te brengen gert@ubuntu:~$ locate allfiles /home/gert/allfiles.txt /home/gert/backups/allfiles.txt.backup

**date**

will display date, time, time zone and more. Check manpage for customization, example :

***date + ‘%A %d-%m-%Y’*** 🡪 Saturday 17-04-2010

***date +%s*** will display you the number of seconds since 1st Jan 1970. When will the conter reach 2 trillion?

***date -d ‘1970-01-01 + 2000000000 seconds’*** 🡪 Wed May 18 04:33:20 CEST 2033

**cal**

displays calendar, use example: ***cal 2 1970*** to display February 1970. Can also show months in the future.

**sleep**

sometimes used in scripts to wait a number of seconds. ***sleep 5***: CLI sleeps for 5 seconds

**time**

stopwatch command, will display how long it takes to do something. Use for example:

***time sleep 5***

**gzip – gunzip**

compresses files, and you can unzip again with ***gunzip***.

### gert@ubuntu:~$ gzip text.txt gert@ubuntu:~$ ls -lh text.txt.gz -rw-rw-r-- 1 gert gert 760K Apr 17 13:11 text.txt.gz gert@ubuntu:~$ gunzip text.txt.gz gert@ubuntu:~$ ls -lh text.txt -rw-rw-r-- 1 gert gert 6.4M Apr 17 13:11 text.txt

**zcat – zmore**

compressed gzip files can be viewed with **zcat and zmore** example: ***zcat text.text.gz***

### gert@ubuntu:~$ zcat text.txt.gz | head -4 / /opt /opt/VBoxGuestAdditions-3.1.6 /opt/VBoxGuestAdditions-3.1.6/routines.sh

**bzip2 – bunzip2**

takes a little more time than ***gzip***, but ccompresses more and better.

**Bzcat – bzmore**

Same thing as zcat, but for this type of compression

**tar (tape archiver)**

This tool archives/extracts a directory structure including ownership and permission settings into single file. Check man tar for extensions. Creating and extracting tar archive from home directory:

### gert@ubuntu:~$ sudo tar -cf /tmp/myhome.tar /home/gert tar: Removing leading `/’ from member names gert@ubuntu:~$ ls -lh /tmp/my\* -rw-rw-r—1 root root 287M okt 19 10:40 /tmp/myhome.tar gert@ubuntu:~$ mkdir myhomebackupfiles && cd myhomebackupfiles gert@ubuntu:~/myhomebackupfiles$ tar -xf /tmp/myhome.tar gert@ubuntu:~/myhomebackupfiles$ tree . └── home    └── gert        ├── count.txt        ├── Desktop        ├── Documents        │   └── School        │       ├── Aardrijkskunde\_oef01        │       └── Semester1        │           └── kwartaal1        │               └── Systems\_Ess        │                   └── Linux        │

### gert@ubuntu:~$ sudo tar -czf /tmp/myhome.tar.gz /home/gert tar: Removing leading `/’ from member names gert@ubuntu:~$ ls -lh /tmp/my\* -rw-rw-r—1 root root 287M okt 19 10:40 /tmp/myhome.tar -rw-rw-r—1 root root 243M okt 19 10:40 /tmp/myhome.tar.gz gert@ubuntu:~$ tar -xzf /tmp/myhome.tar.gz -C /tmp/

### gert@ubuntu:~$ tree /tmp/home /tmp/home └── gert     ├── count.txt     ├── Desktop     ├── Documents     │   └── School     │       ├── Aardrijkskunde\_oef01     │       └── Semester1     │           └── Kwartaal1     │               └── Systems\_Ess     │                   └── Linux     │

A tar file is often compressed with an option for the tar command. Can be done with gzip. The compressed tar file is called a **tarball.** So when asked on exam to make a tarball, something needs to be compressed.

There is a difference between using a relative path and an absolute path. The leading slash is always removed, but with an absolute path the whole path is put in the tar file.

### gert@ubuntu:~$ cd /home gert@ubuntu:/home$ ls gert gert@ubuntu:/home$ sudo tar -czf /tmp/myhome\_relative.tar.gz gert tar: Removing leading `/’ from member names gert@ubuntu:~$ ls -lh /tmp/my\* -rw-rw-r—1 root root 243M okt 19 10:40 /tmp/myhome\_relative.tar.gz -rw-rw-r—1 root root 287M okt 19 10:40 /tmp/myhome.tar -rw-rw-r—1 root root 243M okt 19 10:40 /tmp/myhome.tar.gz gert@ubuntu:~$ mkdir myhomerestore && cd myhomerestore gert@ubuntu:~/myhomerestore$ tar -xzf /tmp/myhome\_relative.tar.gz gert@ubuntu:~/myhomerestore$ tree . └── gert     ├── count.txt     ├── Desktop     ├── Documents     │   └── School     │       ├── Aardrijkskunde\_oef01     │       └── Semester1     │           └── Kwartaal1     │               └── Systems\_Ess     │                   └── Linux

Notice that there is no directory “home” anymore. This is why it is better to use an absolute path if you want to restore to the same directories.

**Chapter 19 – Regular Expressions**

**Regex versions**

Three different versions of regular expression syntax, one or more of these syntaxes can be used.

**BRE:** Basic Regular Expressions

**ERE:** Extended Regular Expressions

**PRCE:** Perl Regular Expressions

Example: ***grep*** tool has the **-E** option to force a string to be read as **ERE**, while **-G** forces **BRE**, and **-P** forces **PRCE**. ***grep*** also has **-F** to force the string to be read literally.

**grep**

Good example of a regular expression. ***grep a***will look for contents in file containing a as character.

**concatenating characters**

Two concatenated characters will have to be concatenated in the same way to have a match. Eg: ***grep ia*** will return name Tania from test file.

**one or the other**

**PRCE** and **ERE** both use | to signify OR. Eg: grep for lines containing letter i or letter a: ***grep -E ‘i|a’ test.txt***

*Or:*

***grep -e ‘i' -e ‘a’ test.txt***  to search for all patterns

**BRE vs ERE**

In BRE the meta-chars ?, + , {, |, ( and ) lose special meaning. You need to use the backslashed version to interpret right, like: \?, \+, \{, \|, \( and \).

### gert@ubuntu:~$ grep 'i|a' list

### gert@ubuntu:~$ grep 'i\|a' list Tania Laura

**zero, one or more**

The \* signifies **zero, one or more** occurrence of the previous character, the **+** signifies one or more:

### gert@ubuntu:~$ grep 'o\*' list2 ll lol loool

### gert@ubuntu:~$ grep -E 'o+' list2 lol loool

**a dot**

can signify any character:

### gert@ubdesk:~$ grep 't.k' chickens tik tak tok

**match the end of a string**

use a dollar character to match the end of a string, and use ^ to match the start of the string:

### gert@ubuntu:~$ grep r$ names Fleur Floor gert@ubuntu:~$ grep ^Val names Valentina

these sigs are called anchors in a regex.

**Separating words**

When grepping, use ***\b*** to find only searched words, or ***-w:***

### gert@ubuntu:~$ grep over text The governer is governing. The winter is over. Can you get over there? gert@ubuntu:~$ grep '\bover\b' text 🡪 OR *grep -w over text* The winter is over. Can you get over there?

**Preventing shell expansion of a regex**

It is advised to always quote the regex, this prevents shell expansion. Example: ***grep ‘r$’ names***:

### gert@ubuntu:~$ cat students | grep Bea Custers grep: Custers: No such file or directory

### gert@ubuntu:~$ cat students | grep "Bea Custers" 1A - Bea Custers

**rename**

the **/usr/bin/rename** is ***perl*** script.

**Perl**

The rename command is a perl script that uses regular expressions. The complete manual for these can be found by typing perldoc perlrequick after installing perldoc with sudo apt install perldoc.

**Well known syntax**

The most common use of the rename is to search for filenames matching a certain string and replacing this string with another string.

### gert@ubuntu:~$ ls abc       allfiles.TXT gert@ubuntu:~$ rename 's/TXT/text/' \* can also do rename 's/text/txt/' \*.text gert@ubuntu:~$ ls abc       allfiles.text

**a global replace**

You enter a regex between the first two slashes, and a replacement between the last two. S signifies switch and g stand for global. The example used the -n switch to show what is being done.

### gert@ubuntu:~$ rename -n 's/TXT/txt/g' aTXT.TXT aTXT.TXT renamed as atxt.txt

**case insensitive replace**

the char i replaces a case insensitive string with another string: **file1.text  file2.TEXT  file3.txt  
gert@ubuntu:~/files$ rename 's/\.text/.txt/i' \*  
file1.txt  file2.txt  file3.txt**

**renaming extensions**

example to rename the file extension only: use a dollar sign:

### allfiles.txt  bllfiles.txt  cllfiles.txt  really.txt.txt  temp.txt  tennis.txt gert@ubuntu:~$ rename 's/\.txt$/.TXT/' \* allfiles.TXT  bllfiles.TXT    cllfiles.TXT    really.txt.TXT temp.TXT      tennis.TXT

without the $ sign the command would fail on the really.txt.txt file.

**sed**

**stream editor**

uses regex for stream editing. Example:

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

**interactive editor**

**sed** can also be used in-place on a file.

### gert@ubuntu:~/files$ echo Sunday > today gert@ubuntu:~/files$ cat today Sunday gert@ubuntu:~/files$ sed -i 's/Sun/Mon/' today gert@ubuntu:~/files$ cat today Monday

**simple back referencing**

& can be used to reference searched and found string

The & is used to double the occurrence of the found string.

### gert@ubuntu:~$ echo It is my Sun | sed 's/Sun/great &day/' It is my great Sunday gert@ubuntu:~$ echo Sunday | sed 's/Sun/&&/' SunSunday

**back referencing**

() are used to group sections so they can later be referenced:

### echo Sunday | sed 's\_\(Sun\)\_\1ny\_' 🡪 Sunnyday

### echo Sunday | sed 's\_\(Sun\)\_\1ny \1\_' 🡪 Sunny Sunday

**multiple back referencing**

each paranthese can be referenced separately by consecutive numbers: also with -r for ERE, called grouping.

### gert@ubuntu:~$ echo 2024-04-01 | sed 's/\(....\)-\(..\)-\(..\)/\1+\2+\3/' 2024+04+01 gert@ubuntu:~$ echo 2024-04-01 | sed -r 's/(....)-(..)-(..)/\3:\2:\1/' 01:04:2024

**white space**

the \s can refer to white space or tab. Example replaces multiple spaces with 1 space.

### gert@ubuntu:~$ echo -e 'today\tis\twarm' today   is      warm gert@ubuntu:~$ echo -e 'today\tis\twarm' | sed 's\_\s\_ \_g' today is warm

**optional occurrence**

a question mark wignifies that the previous character is optional.

### gert@ubuntu:~$ cat list2 | sed -r 's/ooo?/A/' ll lol lAl lAl lAol

**exactly n times**

You can demand an exact number of times the previous char has to occur. Eg: exactly 3 o’s.

### grep -E 'o{3}' list2 🡪 loool

### cat list2 | sed -r 's/o{3}/A/' 🡪lAl

**between n and m times**

eg: ***grep ‘lo\{2,3\}*|/gone/’ 🡪 lool, loool 🡪 gone, gone**

**bash history**

the bash shell can also interpret some regular expressions. How to manipulate exclamation mask history feature of the bash shell:

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

**Chapter 21 – Users**

This chapter covers how to identify your user account on a Unix computer

**whoami**

tells you your username

**who**

will give you information about who is logged on to the system

**w**

shows you who is logged on and what they are doing

**id**

gives you userid, primary group id, and a list of the groups that you belong to

**su to another user**

allows you to run a shell as another user. Example:

### laura@ubuntu:~$ su tania Password: tania@ubuntu:/home/laura$

**su to root**

The root user can become any existing user without knowing that persons password.

**su –**

When no username is provided to ***su***  or ***su -***, the command will assume root is the target. This only works if the root user has a password.

**run a program as another user**

Before ***su*** works, the system admin has to set up the /etc/sudoers file. This can be useful to delegate administrative tasks to another user without giving the root password.

### gert@ubuntu:~$ sudo /usr/sbin/useradd -m valentina [sudo] password for gert: gert@ubuntu:~$

**visudo**

Editing the sudoers file. Out of scope for this course.

**Sudo su –**

to switch user to root. All users can run admin commands with su in front of command. With sudo su the user becomes root.

**sudo logging**

Using sudo without authorization will result in a severe warning.

**user management**

User management can be done in three complementary ways. Either:

* By means of **graphical** tools provided by your distribution. User friendly and good for beginners.
* By use of **command line tools** like useradd, usermod, gpasswd, passwd, and others.
* You can also **edit the local configuration files** directly using vi.

**etc/passwd**

The local user database on Linux is /etc/passwd. It contains seven coloumns separated by a colon. Makeup:  
  
username : x : userid : primary group id : description : name of home directory : login shell

More info to be found with ***man 5 passwd***

**root**

The root is the most powerful account on your Linux system. Can do almost anything, including creation of other users. Root user always has userid 0, regardless of the name of the account.

**useradd**

can add users with the useradd command. Example of how to add user named ‘yanina’, and at the same time forcing the creation of the home directory (-m), setting the name of the home directory:

### root@ubuntu:~# useradd -m -d /home/yanina -c "yanina wickmayer" yanina root@ubuntu:~# tail -1 /etc/passwd yanina:x:529:535:yanina wickmayer:/home/yanina:/bin/bash

**/etc/default/useradd**

This file contains default user options. Besides using cat to display file, you can also use ***useradd -D***.

**userdel**

with ***userdel -r*** it will remove the user + the home directory.

**usermod**

Can modify properties of user with usermod command.

### root@ubuntu:~# tail -1 /etc/passwd harry:x:1003:1003:harry potter:/home/harry:/bin/bash root@ubuntu:~# usermod -c 'wizard' harry root@ubuntu:~# tail -1 /etc/passwd harry:x:1003:1003:wizard:/home/harry:/bin/bash

**creating home directories**

The easiest way to create a home directory is to supply -m option with useradd, like mentioned above. Another less easy way is to create a home directory manually with mkdir, which also requires setting owner and permission of the directory with chmod and chown.

**/etc/skel/**

When using **useradd -m** option, the /etc/skel/ directory is copied to the newly created home dir. /etc/skel contains some usually hidden files that contain profile settings and default values for applications. In this way /etc/skel serves as a default home directory and as a default user profile.

**Deleting home directories**

The -r extension of userdel will delete home directory together with user account.

**Login shell**

The /etc/passwd file specifies the login shell for the user. Different users can have different login shells. Can use usermod to change the shell for a user.

### root@ubuntu:~# tail -2 /etc/passwd annelies:x:527:533:sword fighter:/home/annelies:/bin/bash laura:x:528:534:art dealer:/home/laura:/bin/ksh root@ubuntu:~# usermod -s /bin/bash laura root@ubuntu:~# tail -1 /etc/passwd laura:x:528:534:art dealer:/home/laura:/bin/bash

**chsh**

Change shell. Use cat /etc/shells to get an overview of the available shells.

### [laura@centos7 ~]$ chsh -s /bin/ksh Changing shell for laura.

**Passwd**

Use this command to change passwords. Upon typing in the old pswd first, new one will be set afterwards.

**Shadow file**

This file is read only and can only be read by root. Users an change their password with the usr/bin/passwd command. The etc/shadow file contains nine colon separated coloumns. Makeup:

Username : encrypted passwd : the day the paswd was last changed :

number of days paswd must be left unchanged : password expiry day : warning number of days before pass expiry :

number of days after expiry before disabling account : the day the account was disabled : no meaning yet

**encryption with passwd**

Passwords are stored in an encrypted format. The encryption is done by the crypt function. This happens automatically when a user is added with useradd -m, and then setting the passwd for that user by doing passwd ‘username’

### root@ubuntu:~# useradd -m xavier root@ubuntu:~# passwd xavier Changing password for user xavier.

**etc/login.defs**

This file contains default settings for user passwords like password aging and length settings. Can look into it by grepping for example: ***grep PASS /etc/login.defs***.

**Chage**

Can be used to set an expiration date for a user account (-E), set a minimum (-m), and maximum (-M) password age, a password expiration date, and set number of warning days before passwd expiration date. The -l option will list these settings for a user.

**disabling a password**

passwords in /etc/shadow cannot begin with an exclamation mark. It is possible to lock or disable an account by typing a ! as the first letter for somebody’s password. Example of disabling somebody’s account, and unlocking it again:

### root@ubuntu:~# grep laura /etc/shadow | cut -c1-70 laura:$6$JYj4JZqp$stwwWACp3OtE1R2aZuE87j.nbW.puDkNUYVk7mCHfCVMa3CoDUJV root@ubuntu:~# usermod -L laura

### root@ubuntu:~# usermod -U laura root@ubuntu:~# grep laura /etc/shadow | cut -c1-70 laura:$6$JYj4JZqp$stwwWACp3OtE1R2aZuE87j.nbW.puDkNUYVk7mCHfCVMa3CoDUJV

**editing the local files**

You can use **vipw** instead of **vim** directly. The vipw tool will do proper locking of the file

### root@ubuntu:~# vipw /etc/passwd vipw: the password file is busy (/etc/ptmp present)

**system profile**

Both the bash and ksh shell will verify the existence of /etc/profile and source if it exists. When reading this script, you will notice that it builds the PATH environment variable, and it might also change the PS1, set the hostname, and execute more scripts.

**~/.bash\_profile**

When this file exists in the home directory, bash will source it. It also adds $HOME/bin to the $PATH variable.

**~/.bash\_login**

When bash\_profile doesn’t exist, bash will check for bash\_login and source it.

**~/.profile**

When neither bash\_profile or bash\_login exist, then bash will verify existence of profile and execute it.

**~/.bashrc**

This script is often sourced by other scripts. On Red Hat bashrc checks for /etc/bashrc and sources it, it also leaves room for custom aliases and functions. On Debian the script is a lot longer, and configures PS1, together with some history variables and a number of active/inactive aliases.

**~/.bash\_logout**

When exiting bash, it can execute **~/.bash\_logout**, Debian uses this opportunity to clear the console screen.

### serena@ubuntu:~$ cat .bash\_logout # ~/.bash\_logout: executed by bash(1) when login shell exits.

**Chapter 22 – Groups**

**Groupadd**

Groups are added with this command. Example: ***groupadd tennis, groupadd salsa, etc…***

**group file**

Users can be a member of several groups. Group membership is defined by /etc/group file.

### root@ubuntu:~# tail -5 /etc/group tennis:x:1006:

The first field is the groups name, second field is groups encrypted passwd, third field is group id or ***GID***, the fourth field is list of members, the group above has no members.

**groups**

A user can type the ***groups*** command to see a list of groups where the user belongs to. Simply type groups.

**Usermod to add users to a secondary(supplementary group)**

Group membership can be modified with useradd or usermod. By default the usermod command will remove the user from every group if the group is not listed in the command! Using -a(append) prevents this behaviour.

### root@ubuntu:~# usermod -a -G tennis inge root@ubuntu:~# usermod -a -G tennis katrien root@ubuntu:~# usermod -a -G salsa katrien root@ubuntu:~# tail -5 /etc/group tennis:x:1006:inge,katrien salsa:x:1010:katrien

**usermod to specify a primary (login) group for a user**

The primary group will become the groupowner of every file or folder you create. Can also be changed with usermod command. You will change the primary group when using usermod with lower letter -g. This will only make a modification in the passwd file, nothing will be changed in the group file.

**Gpasswd**

Group membership can also be modified with the gpasswd command. With this command you can add a user to a group. You can also remove a user from a group

### root@ubuntu:~# gpasswd -a gert tennis Adding user gert to group tennis

### root@ubuntu:~# gpasswd -d gert tennis Removing user gert from group tennis

**groupmod**

You can change the group name with groupmod:

### root@ubuntu:~# groupmod -n darts snooker

**groupdel**

Permanently delete group with the groupdel command

### root@ubuntu:~# groupdel tennis

**gpasswd**

You can delegate control of group membership to another user with the gpasswd command. Group admins do not need to be part of the group. They can remove themselves from a group, but this does not influence their ability to add or remove members. Informationabout group admins is kept in the /etc/gshadow file. To remove all group admins from group, use gpasswd to set an empty admin list:

### root@ubuntu:~# gpasswd -A serena sports root@ubuntu:~# su - serena serena@ubuntu:~$ id harry uid=516(harry) gid=520(harry) groups=520(harry) serena@ubuntu:~$ gpasswd -a harry sports Adding user harry to group sports root@ubuntu:~# gpasswd -A "" sports

**newgrp**

You can start a child shell with a new temporary primary group using the newgrp command.

### root@ubuntu:~# mkdir prigroup root@ubuntu:~# cd prigroup/ root@ubuntu:~/prigroup# touch standard.txt root@ubuntu:~/prigroup# ls -l total 0 -rw-r--r--. 1 root root 0 Apr 13 17:49 standard.txt root@ubuntu:~/prigroup# echo $SHLVL 1 root@ubuntu:~/prigroup# newgrp tennis root@ubuntu:~/prigroup# echo $SHLVL 2 root@ubuntu:~/prigroup# touch newgrp.txt root@ubuntu:~/prigroup# ls -l total 0 -rw-r--r--. 1 root tennis 0 Apr 13 17:49 newgrp.txt -rw-r--r--. 1 root root   0 Apr 13 17:49 standard.txt root@ubuntu:~/prigroup# exit exit root@ubuntu:~/prigroup# echo $SHLVL 1 root@ubuntu:~/prigroup#

**vigr**

Similar to vipw, vigr can be used to manually edit the /etc/group file, since it will do proper locking of the file.

**Chapter 23 – Standard file permissions**

**File ownership**

The users and groups of a system can be locally managed in /etc/passwd and /etc/group. These users and groups can own files. Every file has a user owner, and a group owner. Use ***ls – lh*** to see all groups and owners of all things.

**listing user accounts**

You can use  ***cut -d: -f1 /etc/passwd | column*** to list all local user accounts.

**chgrp**

You can change the group owner of a file using the **chgrp** command:

### root@ubuntu:/home/gert/owners# ls -l file2 -rw-r--r-- 1 root tennis 185 Apr  8 18:46 file2 root@ubuntu:/home/gert/owners# chgrp snooker file2 root@ubuntu:/home/gert/owners# ls -l file2 -rw-r--r-- 1 root snooker 185 Apr  8 18:46 file2

**chown**

The user owner of a file can be changed with this command. You can also use chown to change both the user owner, and the group owner.

**list of special files**

When you use ls -l for each file you can see ten chars before the user and group owner. The first char tells us the type of file. Regular files get a -, directories get a d, symbolic links are shown with an l, pipes get a p, character devices get a c, block devices a b, and sockets an s.

### gert@ubuntu:~$ ls -ld /dev/console /dev/sda crw-------   1 root root  5, 1 Mar 15 12:45 /dev/console brw-rw----   1 root disk  8, 0 Mar 15 12:45 /dev/sda

**permissions**

**rwx**

|  |  |  |
| --- | --- | --- |
| 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) |

**Three sets of rwx**

We already know that the output of ls -l starts with ten chars. When you are the **user owner** of a file, then the user owner permissions apply to you. The rest of the permissions have no influence on your access to the file. When you belong to the **group** that is the **group owner** file, then the group owner permissions apply to you.

When you are not user owner, and don’t belong to the group owner, then the other permissions apply to you.

|  |  |  |
| --- | --- | --- |
| Position | Characters | Function |
| 1 | - | This is a regular file |
| 2-4 | Rwx | Permissions for the user owner |
| 5-7 | r-x | Permissions for the group owner |
| 8-10 | r-- | Permission for others |

**Example:**

### drwxr-xr-x 2 gert gert 4.0K 2007-02-07 22:26

**setting permissions (chmod)**

Permissions can be changed with chmod.

***Example1:*** gives the user owner execute permissions:   
-rw-r--r-- 1 gert gert 0 2007-02-07 22:34 permissions.txt  
gert@ubuntu:~/perms$ chmod u+x permissions.txt

***Example2:*** remove group owners read permission:   
**gert@ubuntu**:~/perms$ chmod g-r permissions.txt

***Example3:*** removes the others read permission:   
**gert@ubuntu**:~/perms$ chmod o-r permissions.txt

***Example3:*** gives all write permission: Can use a+w, or just +w  
**gert@ubuntu**:~/perms$ chmod a+w permissions.txt

***Example4****:* set explicit permissions:   
**gert@ubuntu**:~/perms$ chmod u=rw permissions.txt

***Example5 :*** Can make combinations :   
**gert@ubuntu**:~/perms$ chmod u=rw,g=rw,o=r permissions.txt

or:

**gert@ubuntu**:~/perms$ chmod u=rwx,ug+rw,o=r permissions.txt

**setting octal permissions**

|  |  |  |
| --- | --- | --- |
| **Binary** | **Octal** | **Permission** |
| 000 | 0 | --- |
| 001 | 1 | --x |
| 010 | 2 | -w- |
| 011 | 3 | -wx |
| 100 | 4 | r-- |
| 101 | 5 | r-x |
| 110 | 6 | rw- |
| 111 | 7 | rwx |

This makes 777 = rwxrwxrwx, and 654 = rw-r-xr—

Chmod will accept these numbers. Example: ***chmod 777 permissions.txt***

**umask**

When creating a file or directory, default permissions are applied. These default permissions are determined by the umask. The umask specifies permissions that you do not want set on by default.

**gert@ubuntu:~$** umask  
0002  
gert@ubuntu:~$ umask -S  
u=rwx,g=rwx,o=rx  
gert@ubuntu:~$ mkdir testdir  
gert@ubuntu:~$ ls -ld testdir  
drwxrwxr-x  1 gert gert 0 Jul 24 06:03 testdir  
gert@ubuntu:~$ touch testfile  
gert@ubuntu:~$ ls -l testfile  
-rw-rw-r--  1 gert gert 0 Jul 24 06:03 testfile

As seen, the file is not executable by default. You have to explicitly do chmod + x to make a file executable.

**mkdir -m**

-m sets the mode.

**gert@ubuntu:**~$ mkdir -m 700 MyDir  
gert@ubuntu:~$ mkdir -m 777 Public  
gert@ubuntu:~$ ls -dl MyDir/ Public/  
drwx------ 2 gert gert 4096 2011-10-16 19:16 MyDir/  
drwxrwxrwx 2 gert gert 4096 2011-10-16 19:16 Public/

**cp -p**

To preserve permissions and time stamps from source files, use cp -p

gert@ubuntu:~/perms$ sudo cp file\* cp  
gert@ubuntu:~/perms$ sudo cp -p file\* cpp  
gert@ubuntu:~/perms$ ls -l \*  
-rwx------ 1 gert gert 0 2018-08-25 13:26 file33  
-rwxr-x--- 1 gert gert 0 2018-08-25 13:26 file42  
  
cp:  
total 0  
-rw-r--r-- 1 root root 0 2020-11-14 15:02 file33  
-rw-r--r-- 1 root root 0 2020-11-14 15:02 file42  
  
cpp:  
total 0  
-rwx------ 1 gert gert 0 2018-08-25 13:26 file33  
-rwxr-x--- 1 gert gert 0 2018-08-25 13:26 file42