

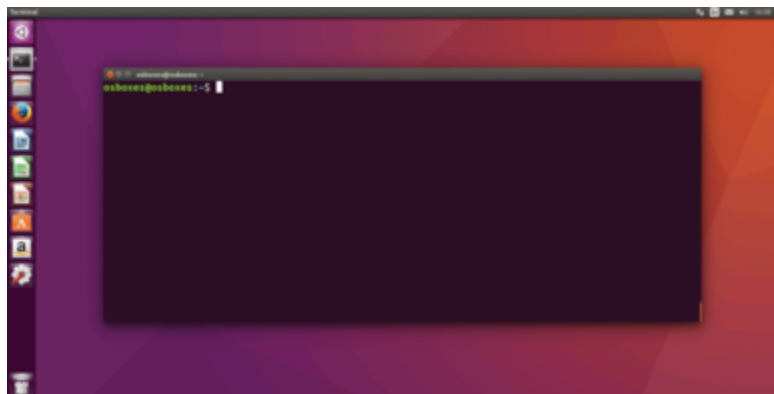
An overview of the most important Linux commands

Like most modern operating systems, Linux has two available interfaces for user input. All of the settings that you set via the graphical user interface (GUI) can also be made in the form of **command line directives** via the so-called **shell**.

Shell is a program that **functions as an interface between system and user**. It includes a command line interpreter that accepts user input via the keyboard, evaluates them, starts programs (if necessary), and returns the output in the form of a text entry to the user. In addition, each shell has its own programming language which makes it possible to write shell scripts – for example, to link program calls and facilitate administrative tasks.

Each shell runs in a **terminal**. At the beginning of the computer age, independent devices, or so-called hardcopy terminals (printer or screen plus keyboard), were used. These were replaced on modern computers by **terminal emulators** – programs that provide users with a graphical window for interacting with the shell.

As soon as you access the terminal of your operating system, it starts the standard shell (i.e. the *Bourne again shell*, Bash) specified in the settings, and accepts input at the **prompt**.



(fileadmin/DigitalGuide/Screenshots/ubuntu-bash.png)

The Bash under Ubuntu: The prompt shows the username and host name

Over time, various shells for unix-like operating systems have been developed which differ in terms of functionality and user-friendliness. As a Linux user, you have the choice of which command line interpreter to use. On most operating systems, in addition, the `dash` and `zsh` shells are already installed. The switch from one shell to another can be comfortably carried out from the terminal (see `chsh` in the user account management chapter). Among the most popular shells, beside the standard programs *Bash* and *Dash*, are *Fish*, *Z-shell*, *Korn-shell*, *(t)csh*, and *Mksh*.

Contents

1. Basic commands (server/configuration/linux-commands-an-overview-of-terminal-commands/#c94060)
2. Help pages (server/configuration/linux-commands-an-overview-of-terminal-commands/#c94062)
3. Directory operations (server/configuration/linux-commands-an-overview-of-terminal-commands/#c94064)
4. File operations (server/configuration/linux-commands-an-overview-of-terminal-commands/#c94066)
5. Rights management (server/configuration/linux-commands-an-overview-of-terminal-commands/#c94068)
6. Search options (server/configuration/linux-commands-an-overview-of-terminal-commands/#c94070)
7. User information (server/configuration/linux-commands-an-overview-of-terminal-commands/#c94072)
8. User account management (server/configuration/linux-commands-an-overview-of-terminal-commands/#c94074)
9. System commands (server/configuration/linux-commands-an-overview-of-terminal-commands/#c94076)
10. System information (server/configuration/linux-commands-an-overview-of-terminal-commands/#c94079)
11. Hardware information (server/configuration/linux-commands-an-overview-of-terminal-commands/#c94081)
12. Process management (server/configuration/linux-commands-an-overview-of-terminal-commands/#c94083)
13. Pager (server/configuration/linux-commands-an-overview-of-terminal-commands/#c94085)
14. Editors (server/configuration/linux-commands-an-overview-of-terminal-commands/#c94087)
15. Network management (server/configuration/linux-commands-an-overview-of-terminal-commands/#c94089)
16. Archive and compress (server/configuration/linux-commands-an-overview-of-terminal-commands/#c94091)
17. Partition management (server/configuration/linux-commands-an-overview-of-terminal-commands/#c94094)
18. Miscellaneous (server/configuration/linux-commands-an-overview-of-terminal-commands/#c94096)

Interaction with the shell usually happens via commands, which can be used to call command line programs of the same name. For every action that you want to carry out via the terminal, use a program call following this basic scheme:



(/digitalguide/)

1 | BEFEHL [OPTIONEN] [ARGUMENTE]

A program call via the terminal uses the name of the program. Most programs offer the possibility to address certain program functions via **options**. If a program expects **arguments** – i.e. in the form of files or index paths – these are usually specified according to the selected options.

In the following sections, we offer an overview of the **most common Linux commands** and their associated command line programs.

Basic commands

In the basic commands category, you'll find the Linux basic commands that are used to **control the terminal**. Learn how to clear the terminal's visibility, retrieve previous terminal entries from the history, or exit the terminal session.

Command	Description
clear	Clear terminal
	Use the command line directive <i>clear</i> to clear the screen content.
	<i>clear</i>
	You'll receive a blank terminal with a prompt. Older entries remain in the scrollback buffer.
	Instead of using this command, you can also clear the terminal with the key combination [Ctrl]+[L].



The command `exit` and line directive `exit` end the current session and closes the terminal.

exit

exit

Instead of this, you can use the key combination [Ctrl]+[D].

View list of all shell commands

Use the *help* command to see a list of all integrated shell commands (built-in commands).

help

help

Call *help* in combination with a shell command to retrieve a short description of the demand in question.

help COMMAND



In Bash, the last 500 commands entered in the (/digitalguide/) command line are saved in the history.

This function serves as entry assistance, and allows you to look through the list of previous commands using the arrow keys and press ENTER again to confirm.

The history can be searched using keywords with the key combination [Ctrl]+[R]. You also have the option to view the complete list, numbered in the terminal.

Use the command *history* without options and arguments.

history

If you want to filter the results, combine *history* via Pipe with the command line program *grep* (see search options) and a keyword.

history | *grep* *KEYWORD*

history

Help pages

You don't know what to do? No worries. Under Linux, there are various **help and documentation pages** available directly via the terminal, such as the *Unix man-pages* and *GNU info pages*. These contain a detailed description of all command line programs, system calls, configuration files, file formats, and core functions. With *whatis* and *apropos*, you can find command line programs in the help pages category, which allow you to search the manual pages of your operating system for keywords.

Command	Description



Use *apropos* to search the page titles and descriptions of your operating system's manual by keywords. The command line program gives you all matches, including a brief description in the terminal.

apropos

Refer to the following scheme:

apropos [OPTIONS] KEYWORD

The command supports different options. Use the *-e* (*--exact*) option to limit the search to exact matches, or use wildcards (*-w '*KEYWORD'*) and regular expressions (*-r*).

Call the GNU info page

Via the *info* command, you can retrieve the GNU info pages for a specific topic. In most cases, these pages correspond to the manual pages that can be accessed via *man*, but as opposed to these, they have links that make the navigators in the manual simpler to read.

Use the following syntax to call a GNU info page:

info [OPTION] TOPIC

A call without an option or topic leads you to the main menu of the GNU info page.

info

Call the manual

The *man* command opens the manual pages (*man-pages*) of your Linux distribution directly in the terminal.

Use the following scheme to call the manual pages:

man [OPTION] TOPIC

The Linux man-pages are divided into 10 topic areas:

- (1) User commands

man

For example, if you want to open the manual page for a specific Linux command, use it in combination with the name of the command.

man clear

You can also narrow down the search by specifying the topic area number:

man 1 clear

In both cases, the manual page opens to the command line directive *clear*. Use the [Q] key to close it and return to the prompt in the terminal.

The *apropos* command provides a way to search the Linux manual pages by keyword.





pinfo

With `pinfo` you have a variant of the command line program `info`, which is based on the command line browser `Lynx` and issues information pages with highlighted links.

Use `pinfo` the same way as the `info` command:

`pinfo [OPTIONS] TOPIC`

Search the manual pages by keyword

The command line program `whatis` serves as a keyword search in the manual pages. Call this program with a popular keyword to search your operating system's manual for exact matches. If there is a match, `whatis` gives a brief description in the terminal.

whatis

`whatis [OPTIONS] KEYWORD`

`whatis (-w '*KEYWORD')` also supports placeholders and regular expressions (`-r`).

Directory operations

You'll use Linux commands for directory operations to create, delete, and manage directories on your system through the terminal, as well as navigate the directory tree. The most important command line directives in this category are `cd`, `ls`, `mkdir`, and `rmdir`.

Command

Description



The command line directive *cd* stands for *change directory*, and is used for navigation in the directory tree.

The syntax of the command reads:

cd [OPTION] DIRECTORY

If no target directory is given, then *cd* automatically switches to the user's home directory.

If *cd* is used with a minus symbol after it (-), it changes back to the previous directory.

Execute program in a new root directory

The *chroot* command (short for *change root*) is used to execute a command in a different root directory. For example, *chroot* is used to isolate critical programs from the rest of the file system. In this case, it's referred to as *chroot jail*.

Calling the program requires root privileges, and is based on the following formula:

chroot DIRECTORY COMMAND

cd

chroot



The command line directive *ls* stands for *list* and is used to display the content of a directory (the names of all files and folders found in the given directory).

The syntax of the command reads:

ls [OPTIONS] DIRECTORY

If *ls* is used without a directory entry, then the command lists the content of the current directory.

With the help of additional options, you can define which information is shown and how it's displayed.

Create directory

The command line directive *mkdir* stands for *make directory*, and allows Linux users to make new directories.

Use the following syntax to create a new directory in the current directory:

mkdir [OPTION] DIRECTORYNAME

You can also create multiple directories at the same time by placing the names separated by a space:

*mkdir [OPTION] DIRECTORYNAME1
DIRECTORYNAME2*

If a directory is supposed to be created in a particular target directory, then specify the absolute or relative path to the directory.

mkdir /home/user/Desktop/test

mkdir ../Desktop/test

With both examples, the *test* directory will be created in the desktop directory.

ls

mkdir



With **mkdirhier** you can create entire directory hierarchies with a single command line directive:

mkdirhier

mkdirhier [OPTION]

/home/user/directory1/directory2/directory3

If *directory1* and *directory2* already exist, *mkdirhier* only creates *directory3*. Otherwise, all three directories are created.

Output directory name

pwd

Use *pwd* (short for *print working directory*) to output the name of the current working directory.

The syntax of the command reads:

pwd [OPTIONS]

Delete directory

If you want to delete a particular directory, use the command line directive *rmdir* (*remove directory*) according to the following syntax:

rmdir

rmdir [OPTION] DIRECTORY

You can only delete empty directories with *rmdir*. To delete a directory along with all of its contained files and subfolders, use the command *rm* (*remove*) with the option *-r*.

Warning: *rmdir* doesn't require a confirmation for deletion. Selected directories are irreversibly deleted.



tree

While `ls` only lists the content of a directory, the command line directive *tree* can be used to output the entire directory hierarchy recursively as a tree structure.

The command uses the following syntax:

```
tree [OPTIONS] [DIRECTORY]
```

File operations

The Linux commands in this chart allow you to carry out various file operations from the terminal. Use the Linux basic commands like *cp*, *mv*, and *rm* to copy, move, rename, or delete files on your system.

Command	Description



A file path is passed to the command line directive `basename`, which simply returns the file name without a default path.

The syntax of the command reads:

basename [OPTIONS] path/to/files [SUFFIX]

For example, if you enter \$ *basename/home/user/photo.jpg* in the terminal, you'll receive the following output:

photo.jpg

The additional input of the suffix removes this from the output as well.

Input: \$ *basename/home/user/photo.jpg .jpg*

Output: *photo*

The command can be expanded to multiple files using options.

basename



The core **1&1** and line program *cat* (short for *concatenate*) was developed as a tool for the combination of file content and can be used as a pager for the display of file content in the terminal.

Use *cat* with the following syntax in the terminal to read a file and output it to *stdout* (the standard output):

cat *OPTIONS* *FILE*

Multiple files can be separated by spaces:

cat *OPTIONS* *FILE1* *FILE2*

Linking file content is done with the use of redirection operators (>, <, and |). For example, use the operator “bigger than” (>) to combine the content of two files into a third.

cat *file_1.txt* *file_2.txt* > *file_3.txt*

Align files at byte level

cmp is part of the *diff* package and is used to compare file contents. As opposed to *diff*, the alignment is done at the byte level and so is particularly suitable for binary files.

Use *cmp* according to the following syntax:

cmp [*OPTIONS*] *FILE1* *FILE2*

cmp finds differences, and then the command line program outputs the byte and line number of the first deviation in the terminal.

cat

cmp



Use the **1&1** command line program *comm* to compare sorted files (i.e. via *sort*) line by line.

The program call is based on the following syntax:

comm [OPTIONS] FILE1 FILE2

If *come* is called without options, the program generates an output with three columns: The first column contains all lines that only appear in *FILE1*, the second column contains all lines only in *FILE2*, and the third column contain all lines that appear in both files.

The program supports three options:

- 1: suppress unique lines from *FILE1*
- 2: suppress unique lines from *FILE2*
- 3: suppress all lines contained in both files

Copy files or directories

The command line directive *cp* (*copy*) is used to copy files and directories. The basic syntax of the command reads:

cp [OPTIONS] SOURCE TARGET

The SOURCE is the element that is intended to be copied. Either a file or a directory is then defined as the TARGET of the copying process. If you define an existing file as the target file, its contents are overwritten with the source file. You also have the option to create a new file with whatever name you choose as the target file.

If you want to copy multiple files, then the target has to be a directory. The same goes for copying a directory.

Copy a source file to a target file in a current directory:

comm

cp

Digital Guide *cp [OPTIONS] SOURCEFILE TARGETFILE*



Example: *cp file.txt file_copy.txt*
(/digitalguide/)

Copy a source file from the current directory to a target directory:

cp [OPTIONS] SOURCEFILE TARGETDIRECTORY

Example: *cp file.txt home/user/documents/2017*

Copy multiple source files to a target directory:

*cp [OPTIONS] SOURCEFILE1 SOURCEFILE2
TARGETDIRECTORY*

Example: *cp file.txt file.odt
home/user/documents/2017*

Copy a source directory from the current directory to a target directory:

cp SOURCEDIRECTORY TARGETDIRECTORY

Example: *cp directory1 home/user/documents/2017*

If a directory is meant to be copied along with all its contents, then all subdirectories need to be included in the copy process using OPTION *-R*.

Extract file content

The *cut* command allows you to extract the contents of a file from the text line of a file (i.e. log or CSV files).

The syntax of the command reads:

cut [OPTIONS] FILE

The exact position of an extracted section is defined via the options *-b* (byte position), *-c* (character position), *-d* (delimiter), and *-f* (field).



The command line program *diff* serves to compare two files. You can also use *diff* to determine if two directories contain the same files.

diff

Call the program to the terminal using the following syntax:

```
diff [OPTIONS] FILE1 FILE2
```

Output file path

dirname is the counterpart to *basename*. The command line directive allows you to extract the path portion from a file path and output it in the terminal without file names.

dirname

The syntax of the command reads:

```
dirname [OPTIONS]
```

For example, enter `$ dirname/home/user/photo.jpg` into the terminal to get the following output:

```
/home/user
```

Output file type

With the command line directive *file* you can output information about the file type of a file.

file

The call is based on the following syntax:

```
file [OPTIONS] FILE
```



The command line program *ln* (short for *link*) generates a shortcut to a file or a directory. This creates another directory entry for this file, which allows you to access the respective file via another file path.

The call for the command line program has to always contain at least the path to the source file.

ln [OPTIONS] path/to/sourcefile

In this case, a shortcut will be created in the current work directory under the same name.

You can also enter a target path and then name the shortcut however you want:

ln [OPTIONS] path/to/sourcefile path/to/shortcut

By default, *ln* creates so-called hardlinks. These aren't suitable for creating shortcuts to directories. Hardlinks also can't be used beyond partition boundaries. So the command is often used with OPTION *-s* (*--symbolic*), with which symbolic links can also be created beyond file system boundaries. Symbolic links also point to and depend on a "real" file path.



ls stands for *list open files*, a tool that gives you information about open files in the terminal, sorted by PID (process ID).

Isof

Call the program to the terminal using the following syntax:

Isof [OPTIONS]

Since unix-like systems such as Linux generally follow the policy that “Everything is a file,” the list outputted by the *Isof* command is accordingly long. As a rule, the options are used to limit this output.

md5sum

Calculate checksums

The command line directive *md5sum* helps you calculate and check MD5 checksums for files.

mv

Move file or directory

The command line program *mv* (*move*) copies a file or directory and deletes the original element. If it’s used within the same directory, then *mv* can be used to rename files.

The program call is based on the following syntax:

mv [OPTIONS] SOURCE TARGET

Application examples:

Move a file to another directory:

mv [OPTIONS] SOURCEFILE TARGETDIRECTORY

Example: *mv file1.txt home/user/documents/2017*

Move multiple source files to a target directory:

*mv [OPTIONS] SOURCEFILE1 SOURCEFILE2
TARGETDIRECTORY*

Example: *mv file1.txt file2.txt*
Digital Guide *home/user/documents/2017*



1&1
(/digitalguide/) Move a subdirectory from the current directory to a target directory:

*mv [OPTIONS] DIRECTORYNAME_OLD
DIRECTORYNAME_NEW*

Example: *mv directory1 home/user/documents/2017*

Rename a file in the current directory

mv [OPTIONS] FILENAME_OLD FILENAME_NEW

Example: *mv file1.txt file2.txt*

Rename a subdirectory in the current directory:

*mv [OPTIONS] DIRECTORYNAME_OLD
DIRECTORYNAME_NEW*

Example: *mv directory1 directory2*



Similar to `cat`, the command line program *paste* also enables the output of file contents to the standard output. But while *cat* merely combines content, *paste* joins column by column.

The basic syntax of the command reads:

```
paste [OPTIONS] FILE1 FILE2 ...
```

In standard mode, the listed files are merged so that all rows with the same row number are transferred to the same line in the output. Every line in the output contains content from all input files.

You can customize which separator is used by *paste* with the option `-d`. Tabs are used as the default separator.

A second mode can be activated using the `-s` option (serial). With this, all lines of the first input file are transferred to the first line of the output. The data for all other input files follows in separate output lines, so each line of the output contains the contents of only one input file.

paste



The command line program *rename* enables the renaming of files and folders with the help of Perl-compatible regular expressions (regex). As opposed to *mv*, the *rename* function is suitable for file operations where the names of several files are supposed to be either partially or completely adapted.

Use *rename* according to the following syntax:

```
rename [OPTIONS] 'REGULAR_EXPRESSION' FILE
```

The syntax of regular expressions reads:

```
s/SEARCHPATTERN/REPLACEMENT/MODIFIER
```

The following example renames all *.htm* file endings to *.html*.

```
rename 's/\.html$/\.html/' *.html
```

Delete file or directory

The command line program *rm* (*remove*) permanently deletes files or entire directories.

The program call is based on the following syntax:

```
rm [OPTIONS] FILE
```

or

```
rm [OPTIONS] DIRECTORY
```

If a directory is to be deleted along with all its subdirectories, then use *rm* plus the option *-R* (*--recursive*).

```
rm -R DIRECTORY
```

Multiple files or directories are separated by spaces.

```
rm [OPTIONS] FILE1 FILE2 ...
```

rename

rm



shred is a command line program that enables safe deletion of files. Chosen elements are overwritten in the course of the deletion process and so can't be restored by forensic means.

The overall syntax of the command reads:

shred [OPTIONS] FILE

Use *shred* with the following options to permanently delete a single file:

shred -fuz FILE

The *-f* option forces the delete process, *-z* overwrites the file contents with zeros (default is random data), and then finally *-u* removes the shredded file from the file system, similar to the *rm* command.

Sort file lists and program output

Use the command line directive *sort* to sort file lists and program output numerically, alphabetically, and by row.

The overall syntax of the command reads:

sort [OPTIONS] FILE

The sorting method can be customized using options: For example, numerical (*-n*), random (*-R*), or in reverse order (*-r*).

shred

sort



The command and line directive *split* is used to divide files.
(/digitalguide/)

The underlying syntax reads:

split [OPTIONS] [INPUT [PREFIX]]

The placeholder *INPUT* corresponds to the file that is to be split. The *PREFIX* acts for the names of the participating files. Their name is based on the following pattern:

PREFIXaa, PREFIXab, PREFIXac ...

Output time stamp

The command line directive *stat* (*status*) outputs access and alteration time stamps for selected files and directories.

The general syntax of the command reads:

stat [OPTIONS] FILE

The output format can be customized with the use of options.

split

stat



The command `touch` and line directive *touch* can be used to modify access and alteration time stamps for files. If *touch* is applied to a file that doesn't exist yet, then it is automatically created. So the command is also good for creating empty files.

Use *touch* according to the following pattern:

touch [OPTIONS] FILE

To set the time stamp for a file to the desired date, use the OPTION *-t* along with the time information in the forms *[YY]MMDDhhmm[.ss]*.

Example:

touch -t 1703231037 file.txt

Access and alteration time stamps are now set to March 23, 2017, 10:37. The modification can be restricted to access or time stamps with the options *-a* and *-m*.

If the *touch* command is used without option *-t*, then it uses the current time stamp.

Delete duplicates in file lists and program outputs

The command line directive *uniq* is usually used in combination with *sort* to clean sorted files from duplicate lines.

In the following example, the *sort* command is linked by a pipe (|) to the *uniq* command to first sort a file and then output it without duplicate lines.

sort file.txt | uniq

touch

uniq

It's quite easy to customize **access and ownership rights for files and directories** in the terminal on Linux. The most important command line directives for rights management are *chown* and *chmod*. Group affiliations are managed by the *chgrp* command.

Digital Guide (/digitalguide/)

1&1

Command	Description
chattr	Manage file attributes
	The command line program <i>chattr</i> (short for <i>change attribute</i>) allows you to view files or directories with attributes. An adjustment of file attributes is supported by various file systems (i.e. ext2, ext3, ext4, XFS, ReiserFS, JFS, and OCFS2).
	Use <i>chattr</i> according to the following syntax to set an attribute:
	<i>chattr [OPTIONS] +ATTRIBUTE FILE</i>
	Set attributes can be removed again using this pattern:
	<i>chattr [OPTIONS] -ATTRIBUTE FILE</i>
	For example, set the attribute <i>-i</i> to prevent changes (deletions or modifications) to a file or a directory:
	<i>chattr +i file.txt</i>
	For other attributes and possible options, refer to the <i>chattr</i> program manual.



The command `chgrp` stands for *change group* and is used for the management of group affiliations for files and directories. To be able to use `chgrp` on a chosen file or directory, you have to have owner or root permissions. These are the only groups to which you can belong.

chgrp

`chgrp` is used according to the following syntax:

`chgrp [OPTIONS] GROUP FILE`

or:

`chgrp [OPTIONS] GROUP DIRECTORY`

The option `-R` refers to subfolders and files contained in a directory.

Manage access rights

The command line program `chmod` (short for *change mode*) is used to assign rights in unix-like file systems (i.e. ext2, ext3, ext4, reiser, xfs).

The general syntax of the command reads:

`chmod [OPTIONS] MODE FILE`

or:

`chmod [OPTIONS] MODE DIRECTORY`

chmod

The MODE placeholder stands for the applicable rights mask. You can find out more about how to create such a system and what to pay attention to in our guide on [access rights with chmod \(server/know-how/allocating-directory-rights-with-chmod/\)](#).

With the help of the `-R` option, rights can be assigned recursively to subfolders and files contained in a directory.



Usually the creator of a file or directory is automatically its owner. The command *chown* stands for *change owner* and allows you to modify the owner permissions.

The command is used according to the following syntax:

```
chown [OPTIONS] [USER][:[GROUP]] FILE
```

or:

```
chown [OPTIONS] [USER][:[GROUP]] DIRECTORY
```

To set owner rights for a user or group, there are four possible combinations available.

Owner and group are reset according to the input:

```
chown [OPTIONS] owner_name:group_name file.txt
```

The group is reset according to the input, the user remains unchanged:

```
chown [OPTIONS] :group_name file.txt
```

The owner is reset according to the input, the group remains unchanged:

```
chown [OPTIONS] owner_name file.txt
```

The user is reset according to the input. The group is set to the default group for the logged-in user:

```
chown [OPTIONS] owner_name: file.txt
```

The changes are recursively extended to subdirectories with the help of OPTIONS *-R*.

chown



Isattr

It you would like to display which attributes are set for a file or directory, use the command line directive *Isattr* (short for *list attributes*) according to the following syntax:

```
Isattr [OPTIONS] FILE/DIRECTORY
```

Search options

Linux offers various command line directives for searching through the system directly from the terminal.

Command

find

Description

Search file system

Using *find* opens up a command line program that serves to search through files.

The program call is based on the following syntax:

```
find [OPTIONS] [DIRECTORY] [SEARCHCONDITION]  
[ACTIONS]
```

The specified directory is the starting directory of the search. The command then searches the starting directory and its subdirectories. If no directory is entered, then *find* starts the search from the current working directory.

Options allow you to define search criteria and actions. The default action is preset at *-print*: The output of the complete file names of all search results to the standard output (usually the terminal).

Common search criteria include the file name (*-name FILENAME[SUFFIX]*), a username (*-user USERNAME*), the file size (*-size n[cwbkMG]*), the access time in days



Search by file name uses meta-characters and placeholders. Put these in quotation marks to prevent interpretation by the shell.

Example:

```
find /tmp -name "*.odt" -mtime -3 -size +20k
```

The */tmp* directory is defined as the starting directory. The command line program *find* outputs all files to the standard output that contain the *.odt* file ending, are larger than 20k, and were changed for the last time less than three days ago.

For other options for the *find* search command, refer to the program's manual.

Search text files

With the *grep* command (short for *global regular expression print*), you can search through text files (i.e. log files). Any character strings or regular expressions can be used as a search pattern.

Use *grep* according to the following syntax:

```
grep [OPTIONS] SEARCHPATTERN [FILE(S)]
```

If *grep* encounters a string that corresponds to the search pattern, then the line number along with the file name is output to the terminal.

In general, *grep* is used on all files in the current directory. The option *-r* enables a recursive search into the subdirectories.



The command and line program *locate* also allows you to search for files through the terminal. But as opposed to *find*, instead of searching through the file directory, it searches a specially created and regularly updated database. As a result, *locate* provides results much quicker than *find*.

To search the database for a particular file, *locate* is used according to the following syntax:

locate SEARCHPATTERN

The search pattern can contain meta-characters as placeholders (*). Put these in quotation marks to prevent interpretation by the shell.

In the following example, *locate* outputs all files with the *.png* ending.

locate ".png"*

The *locate* command is case-sensitive. To gloss over the difference between upper-case and lower-case letters in the search, use the option *-i*.

The file */var/lib/locatedb* functions as a database for the indexing of files. This contains a list of all the files of the file system at a particular time, and so must be regularly updated. To do this, use the *updatedb* command.

locate



tre-agrep also is used to search for strings in text files based on search patterns. But unlike *grep*, it's not only exact matches that are output, but vague results are also allowed, such as those with transposed letters or missing characters. The program is based on the TRE library and makes it available in the command line.

The syntax of *tre-agrep* matches that of the *grep* command:

```
tre-agrep [OPTIONS] SEARCHPATTERN FILE(S)
```

Using options, you can define a maximum error allowance. In the following example, a maximum of one deviation is tolerated.

```
tre-agrep -1 'Linux' test.txt
```

tre-agrep outputs all lines of the text file *test.txt* that contain the words defined by the search pattern or which deviate from it by one letter: i.e. *Linus*.

Update file index

A *locate* search only functions properly if the */var/lib/locatedb* file is continuously kept up to date.

The *updatedb* command allows you to manually update the database. Note that you need root permissions to do this:

```
updatedb
```

tre-agrep

updatedb



(/digitalguide/) With the *whereis* command, you can locate the binary code, source code, or manual files of the selected program.

The general syntax of the command reads:

whereis [OPTIONS] PROGRAM

Example:

whereis firefox

For the output, *whereis* writes the path to the found files separated by spaces in the terminal:

*firefox: /usr/bin/firefox usr/lib/firefox
usr/share/man/man1/firefox.1.gz*

Options can be used to limit the search to specific file types or directories.

Identify program binary files

If you would like to identify the binary files of a program, use the command *which* with the following syntax to output the path in the terminal.

which [OPTIONS] PROGRAM

Example:

which firefox

Output:

/usr/bin/firefox

In the default mode, *which* outputs the first file it finds. Use the option *-a* to show all files that fulfill the search criteria.

whereis

which



Use the command line programs for the following categories to access **detailed information on the registered users in the system** as well as their groups and processes.

Command	Description
finger	Obtain user information The command line program <i>finger</i> serves to access user information. Use the command in combination with the desired username: <i>finger [options] [USERNAME]</i> The output contains the following information for the given user account: login name, real name, login time, time since last activity (<i>idle time</i>), home directory of the user, login shell, location (<i>office number</i>), mail and telephone number (if given), and the contents of the files <i>.plan</i> , <i>.project</i> , <i>.pgpkey</i> , and <i>.forward</i> in the home directory of the user (if given). Use <i>finger</i> without a username to obtain information about your own account.
	Obtain group affiliations The command <i>groups</i> lists the group affiliations of a selected user account. Use the command line directive according to the following pattern: <i>groups [OPTIONS] [USERNAME]</i> Use <i>groups</i> without a username to list all groups to which your user account belongs.



The command and line directive *id* outputs user and group identifiers of the selected user accounts.

id

id [OPTIONS] [USERNAME]

If you want to identify your own IDs, use the command without a username.

The range of the output can be limited using options.

Obtain information about recently logged in users

Use the command *last* according to the following pattern to view a list of recently logged-in users, including login and logout times.

last [OPTIONS]

last

The corresponding information is obtained from the *wtmp* file under */var/log/wtmp*. If you only want to request information about a particular account, then enter the command line directive with the desired username.

last [OPTIONS] [USERNAME]

Obtain current user and their processes

The command *w* outputs a list of all registered users including all processes that they've executed.

Use *w* in combination with a username to limit the command to just this user:

w [OPTIONS] [USERNAME]

w

Range and format of the output can be customized using options.



The command `who` and `who` outputs detailed information about users registered on the system.

The general syntax of the command reads:

```
who [OPTION] [SOURCEFILE]
```

`who` supports various options with which you can customize the range of the outputted information.

By default, `who` refers to data about currently registers users from the `/var/run/utmp` file.

You have the option to specify one of the following files as the source of the information.

If you would like to obtain information about earlier registrations, use `who` in combination with the source file `/var/log/wtmp`:

```
who [OPTION] /var/log/wtmp
```

For information about unsuccessful registrations, enter the command along with the `/var/log/btmp` file:

```
who [OPTION] /var/log/btmp
```

Obtain your own username

Use the command `whoami` to obtain your own username.

```
whoami [OPTIONS]
```

who

whoami

User account management

Linux provides you with a series of programs with which you can create, delete, and manage user accounts and groups directly from the terminal. An overview of the important **Linux commands for user account management** is put together for you here. You'll also find Linux terminal commands in this category that enable you to access code with other user rights, including the super-user root.



Create a user account

(/digitalguide/)

The simplest option for creating a user account is provided by the command line program *adduser*. This requires root permissions, and is used according to the following syntax:

```
adduser [OPTIONS] USERNAME
```

Use *adduser* without options to automatically create a user ID, home directory, and user group with the same name, in addition to the new user account.

Example:

```
adduser test
```

Terminal output:

```
Adding user 'test' (1001) ...
```

```
Adding new group 'test' (1001) ...
```

```
Adding new user 'test' (1001) with group 'test' ...
```

```
Creating home directory '/home/test' ...
```

```
Copying files from '/etc/skel' ...
```

This is followed by an interactive dialog in which you can define the password and other user information (real name, office number, telephone number, etc.).

This automation can be customized or prevented through additional options.

The perl script *adduser* is based on the low-level program *useradd*, and offers the same functions in a user-friendly form.

adduser



The command `chfn` (short for *change finger*) allows you to customize additional information on a user account, such as the real name, office number, and private or work telephone numbers.

The general syntax of `chfn` reads:

`chfn [OPTION "NEW VALUE"] [USERNAME]`

The command has to be executed with root permissions.

Which user information will receive a new value is defined with the help of the option `-f` (real name), `-r` (office number), `-w` (work phone), and `-h` (private phone).

In the following example, the old office number of `peter23` is overwritten with the value 122.

`chfn -r "122" peter23`

chfn



The command and line directive *chsh* (short for *change shell*) changes the login shell of a chosen user.

The syntax reads as follows:

chsh [OPTIONS] USERNAME

To customize the login shell of a user, use *chsh* with the option *-s*. This directs the path to the desired shell (i.e. */usr/bin/fish*).

Notice: Users without root permissions can't change their own shell. If you would like to change the shell of another user, execute the command with root permissions.

Example:

sudo chsh -s /usr/bin/fish peter23

For the user *peter23* the shell *fish* (*friendly interactive shell*) is defined as default.

Changing the shell doesn't take effect until the user logs out and back in again.

chsh



The command and line program *deluser* deletes all entries for a selected user account from the system account files.

Calling *deluser* requires root permissions and uses the following syntax:

```
deluser [OPTIONS] USERNAME
```

Example:

```
deluser peter23
```

The user account *peter23* is now deleted.

If you would also like to delete all files from the home directory of the user, then use the command with the options *--remove-home*. If you want to delete all user files from the system, use the options *--remove-all-files*.

```
deluser --remove-all-files peter23
```

If you want to pack up the user files before deleting them, use *deluser* in combination with the option *--backup-to* and specify the desired directory.

```
deluser --backup-to /path/to/directory peter23
```

deluser is a perl script that provides the functions of the low-level program *userdel* in a more user-friendly form.

deluser



The command line program *groupadd* is used to create user groups.

groupadd

Use *groupadd* with root permissions according to the following syntax:

```
sudo groupadd [OPTIONS] GROUPS
```

Each newly-created group contains its own group ID (GID). Group IDs between 0 and 99 are reserved for system groups. If you want to define the GID for a new user group for yourself, use the command line directive *groupadd* with the option *-g* (GID).

In the following example, the group *users* is created with the GID 1425:

```
groupadd -g 1425 users
```

If you want to create a system group, use the option *-r* (root).

Delete user group

The command line directive *delgroup* (short for *delete group*) deletes an existing user group.

The general syntax of *delgroup* reads:

```
delgroup [OPTIONS] GROUP
```

delgroup

To execute the command, root permissions are required.

The following call will delete the *users* group:

```
delgroup users
```

Similar to *deluser*, this command is also a perl script that offers the functions of the low-level program *groupdel* in a more user-friendly form.



Names and group IDs (GID) of existing user groups can be customized via *groupmod*.
(/digitalguide/)

groupmod

The command line directive is used with root permissions according to the following syntax:

groupmod *OPTIONS GROUPS*

Use *groupmod* with the option *-g* to customize the GID. Call the command with the option *-n* to overwrite the group name.

Examples:

groupmod -g 1800 users

The GID of the *users* group is set at 1800.

groupmod -n all users

The *users* group is renamed *all*.



The command `newgrp` (short for *new group*) allows registered users to change their current group ID without having to log out and back in.

The general syntax of the command reads:

newgroup [-] [GROUP]

If the *newgrp* command is used with the optional parameter `[-]`, then the group change causes a restart of the user environment – as if the user had logged in again.

Users who use *newgrp* without group specification change to the default group specified under */etc/passwd*.

In principle, a user needs to be a member of the group that they want to change into. Password-protected groups are an exception. If a group is protected by a password, then it's prompted by the terminal before the change.

newgrp



Use the **passwd** command line program *passwd* to change the password of a user or define, check, and change intervals.

The command is based on the following syntax:

```
passwd [OPTIONS] USERNAME
```

If you want to change the password of another user, then you need root permissions.

Use the *passwd* command without a username to change your own password.

```
passwd
```

If the password is supposed to be blocked, use the command *passwd* with the option *-l* (*--lock*).

```
passwd -l USERNAME
```

Other options give you the opportunity to define a duration for the expiration of passwords (*-x*) as well as warning (*-w*) and check intervals (*-i*).

```
passwd -x MAX_DAYS -w WARN_DAYS -i  
INACTIVE_DAYS USERNAME
```

The following example defines for the user *peter24* that the password needs to be reset every 30 days. A warning is issued 5 days before the deadline. If the password isn't renewed after 30 days, then the password expires and the user account *peter24* is blocked after 3 days.

```
passwd -x 30 -w 5 -i 3 peter24
```



The command `sudo` (substitute user do) can set the program call to run with the rights of another user. As a rule, the entry of a password is required for this. The command `sudo` always asks for the password of the calling user.

If the command is entered without a username, then the superuser `root` is set as the target user.

sudo PROGRAMCALL

Administrators have the option to define who can use `sudo` and which program calls are allowed in the `/etc/sudoers` file. A user has to belong to the `sudo` group to be able to use the `sudo` command.

If you want to select a different target user, then use `sudo` with the option `-u` and the desired username.

sudo -u USERNAME PROGRAMCALL

Such a user change is only possible if it's allowed in `/etc/sudoers`.

If you want to permanently change the command to run with administrator rights in the root shell, use `sudo` with the option `-i`.

sudo -i

The `sudo` command is useful because it allows users to execute previously defined commands as root users without having to enter the root password.

sudo



su

The command `su` and `sudo` also allows for a temporary user change to run a program call with the rights of a target user. As opposed to *sudo*, the command is not directly executed. Instead, a change of identity occurs. Instead of asking for the password of the calling user, the target user password is requested. To call programs as a superuser, a user needs the root password of the system. Also unlike *sudo*, *su* can't be restricted to a set of pre-defined program calls set by the administrator.

The general syntax of the command reads:

su [OPTIONS] [USERNAME]

A call without a USERNAME selects *root* as the target user.



The command `usermod` gives you the option to edit previously created user accounts.

Use *usermod* with root permissions according to the following syntax:

usermod [OPTIONS] USERNAME

Which modifications are intended can be defined with the help of options.

Change username (*-l NEW_NAME*):

usermod -l peter24 peter23

Create a new home directory (*-d DIRECTORY*) and move the old files (*-m*):

usermod -d /path/to/directory/peter24 -m peter24

All files from the old home directory will be moved to the new home directory.

Block user (*-L*):

usermod -L peter24

The password of the user *peter24* is blocked.

Include users in groups (*-a*) and maintain all other group memberships (*-G*):

usermod -aG users peter24

Peter is added to the *users* group.

usermod

System commands

In the system commands category, you'll find the basic Linux commands for system control. Use the following commands to restart and shut down the system from the terminal – and control them with a timer, if desired.



Most command line directives for system control must be run with root permissions.
(/digitalguide/)



Command	Description
logger	Create log entries
	With the command line program <i>logger</i> , entries in the system log can be created.
	Use <i>logger</i> according to the following pattern: <i>logger "YOUR MESSAGE"</i> Find the system log under <i>/var/log/syslog</i> .
reboot	Restart the system
	The command line directive <i>reboot</i> causes a restart of the system. To trigger a restart, the command has be executed with root permissions.
	<i>reboot [OPTIONS]</i>



The `rtcwake` command and line directive `rtcwake` allows you to start and shut down the system according to a timer.

The command is based on the following syntax:

```
rtcwake [OPTIONS] [MODE] [Time]
```

Choose a particular mode (-m MODE) for the system to move to at a particular time in seconds (-s TIME IN SECONDS). You also have the option to wake up your system at a precisely defined time (-t UNIXTIME).

Example 1:

```
rtcwake -m standby -s 300
```

The system will be put in standby mode for 5 minutes (300 seconds).

Example 2:

```
rtcwake -m off -t 1490997660
```

The system will be shut down and 'woken up' at the Unix time 1490997660. This corresponds to the following date: 4/1/2017 - 12:01:00 A.M. The Unix time is the number of seconds since 1/1/1940 at 12:00 A.M. Because information is hard to find in Unix, it's recommended to translate to the `date` command (listed below).

```
rtcwake -m off -t $(date -d '20170401 00:01' +%s)
```

rtcwake



The command `shutdown` and `shutdown` can be used by the root user to shut down the system.

The command is based on the following syntax:

shutdown [OPTIONS] [TIME] [MESSAGE]

If you want to induce a shutdown, you have the option to define a time that the system should be turned off. For this, use either a concrete time input (hh:mm) or a countdown (+m).

Other users on the system will get a shutdown message. This can be accompanied by a personal message, if needed.

In the following example, the system will be shut down in 10 minutes:

shutdown +10

If the command `shutdown` is used with the option `-r`, the shutdown of the system is followed by a reboot.

shutdown -r +10

shutdown

System information

In the system information category, we've collected command line programs with which you can **obtain information and status reports**, giving you a comprehensive overview of the state of your system.

Command	Description
---------	-------------



The `coreutils` and `date` outputs the system time including the date.

date [OPTIONS] [OUTPUTFORMAT]

If you want to work with a particular time in the context of a program call (see *rtctime*), define this with the help of the option *-d 'DATE'*. In addition, various options are supported that can transfer date and time data to a desired format.

For example, use the option *+%s* to output a date in Unix time (number of seconds since 1/1/1970 00:00:00 UTC).

Example:

date -d '20170427 11:29' +%s


Output:

1493285340

1493285340 Unix time corresponds to 4/27/2017 - 11:29:00 A.M.

date



Use the  command *df* (*disk free*) according to the following pattern to display the free hard drive space of the attached partitions.

df [OPTIONS]

If the command is used in combination with a particular file, the system only specifies the free space on the partition where the file is located.

df [OPTIONS] [FILE]

The option *-l* (*local*) restricts *df* to the local file system. It also supports options that let you customize the output format. For a readable output, it's recommended to use the option *-h* (*human readable*): i.e. 3K 124M 1G.

df



The program `dmesg` (short for *display message*) outputs core circular buffer messages in the terminal and allows you to localize hardware and driver failures.

Use `dmesg` according to the following pattern:

`dmesg [OPTIONS]`

The `dmesg` output contains all messages of the boot routine, and is accordingly long. The command line program is often used in combination with a pager, like *more*, *less*, or *tail*.

Example:

`dmesg | tail`

The `dmesg` output is delivered to the pager *tail* with the help of the pipe operator (`|`). This leads to only the last 10 messages in the terminal being output.

A combination with the *grep* command makes it possible to do a targeted search through the messages.

Access occupied hard drive space

If you want to know how much hard drive space is occupied by directories on your system, use the command `du` (short for *disk usage*) according to the following pattern:

`du [OPTIONS] [DIRECTORY]`

Specifying a particular directory is optional. The occupied hard drive space is output with the option `-h` for a human-readable format.

dmesg

du



The `top` and `free` outputs the memory usage.
(/digitalguide/)

The general syntax reads:

free [OPTIONS]

As output, you'll get two specifications: *Mem* (*Memory*) and *Swap*.

Mem deals with the physical memory of your system. If this is drained, then Linux outsources part of the data saved in the RAM to the hard drive. This is referred to as *swap space*.

Free also supports the option *-h* for outputting the memory usage in a human-readable format.

Retrieve host name

Use the command *hostname* according to the following pattern to display the DNS names of the system.

hostname [OPTIONS]

Retrieve core information

The command line directive *uname* stands for *unix name* and is used to access system information from the core.

The command supports various options with which the output can be filtered according to the desired information.

uname [OPTIONS]

free

hostname

uname



uptime

It you want to determine how long the system has been running since the last reboot, use the command line directive *uptime* according to the following pattern:

uptime

Access statistics about virtual storage

With the help of the monitoring tool *vmstat*, you can access information about virtual memory, reading and writing procedures on the disc, and CPU activity.

Call *vmstat* according to the following syntax to output the average values since the last system start.

vmstat [OPTIONS]

vmstat also offers a continuous monitoring mode that accesses system values as often as requested in a desired time interval in seconds.

vmstat [Options] [INTERVAL [REPETITIONS]]

Example:

vmstat 4 8

The request takes place in eight passages every four seconds.

If you want to stop the continuous request, use the key combination [CTRL] + [C].

vmstat

Hardware information

Linux commands in this category deliver **detailed information about the hardware components** that form the foundation of your system.

Command	Description
---------	-------------



Use *lscpu* (short for *list cpu*) according to the following pattern to output information about the CPU architecture in the terminal.

lscpu

lscpu [OPTIONS]

For possible options, refer to your operating system's manual.

Output hardware information

The command *lshw* stands for *list hardware* and outputs information about the hardware components in the terminal. The information includes CPU, memory modules, and devices such as sound cards, graphics cards, or drives that are connected to PCI, USB, or IDE interfaces.

lshw

Use *lshw* according to the following syntax:

lshw [OPTIONS]

The command supports various options for customizing the output format (*-html*, *-xml*, *-short*, *-businfo*) as well as the range of information (i.e. *-sanitize* to hide sensitive information).

Output information on PCI devices

Use *lspci* (short for *list pci*) according to the following pattern to output detailed information about PCI devices.

lspci

lspci [OPTIONS]

For possible options, refer to your operating system's manual.



Use *lsusb* (short for *list usb*) to output detailed information about USB devices in the terminal.

lsusb

lsusb [OPTIONS]

For possible options, refer to your operating system's manual.


Process management

On Linux, the instance of a running program is called a process. The following terminal commands are part of the standard repertoire of the process management, and allow you to **supervise all processes on your system easily from the terminal** and control as necessary.

Command	Description
chrt	<p>Request and customize real-time attributes</p> <p>The command line program <i>chrt</i> deals with continuous process controls and makes it possible to identify and customize the real-time attributes (scheduling regulation and priority) of running processes, or execute commands and their arguments with specified real-time attributes.</p> <p>The general syntax of the command reads:</p> <p><i>chrt [OPTIONS] [PRIORITY] PID/COMMAND [ARGUMENT]</i></p> <p>Use <i>chrt</i> without entering a priority and with the option <i>-p</i> to identify the real-time attributes of chosen processes:</p> <p><i>chrt -p PID</i></p> <p>Example:</p> <p><i>chrt -p 1234</i></p>

Digital Guide *chrt* outputs the real-time attribute of the process:
1234.



 (/digitalguide/) The command is usually used with the following pattern to execute a command and its arguments with a particular real-time priority:

```
chrt [OPTIONS] PRIORITY COMMAND [ARGUMENTS]
```

Example:

```
chrt 99 firefox
```

The program Firefox is started with a real-time priority of 99.

If, on the other hand, the real-time priority of a process that's already running is supposed to be adapted, then use the following syntax:

```
chrt -p PRIORITY PID
```


Example:

```
chrt -p 20 1234
```

The real-time priority of the process 1234 is set at 20.

chrt also offers the possibility to set or define the scheduling regulation of running or newly started processes with the help of options.

chrt uses SCHED_RR (Round Robin, explicitly with the option *-r*) as the standard value of the scheduling regulation. This means that all computation-ready processes successively get an allocated CPU time for a certain interval. This is known as a time slice, and specifies how long a process can run until it's replaced by another process. The size of the time slice for a process depends on its priority. Linux offers 140 priority levels for processes (0 = highest priority, 139 = lowest priority). The priority levels 1 to 99 are reserved for processes with real-time priority. User processes

are usually carried out with a priority level from 100 to 135. This corresponds to a *nice* value of -20 to +19 (see [here](#)).  [1&1](#)

Digital Guide (/digitalguide/)

In addition to SCHED_RR, Linux also has SCHED_FIFO (option *-f*) for another scheduling regulation for real-time processes. Like SCHED_RR, SCHED_FIFO works as a first-in/first-out algorithm. This way doesn't use time slices. Processes started with SCHED_FIFO run long enough that they either finish or are displaced by a process with a higher real-time priority. Displaced processes return to the end of the queue.



The `ionice` command and line directive *ionice* is used to influence the priority of a process that uses the I/O interface of the core.

The general syntax of the command reads:

ionice [OPTIONS] COMMAND

To be able to execute *ionice*, you need root permissions.

The command distinguishes between three scheduling classes that are passed on using the *-cZAH* option. Possible values are 1, 2, and 3.

1 = Real time: The I/O action is executed immediately.

2 = Best effort: The I/O action is executed as quickly as possible.

3 = Idle: The I/O action is only executed when no other process is taking I/O time.

The PID of a running process is passed on with the option *-pPID*.

Example:

ionice -c2 -p1234

The scheduling class 2 (best effort) is passed to the process with the PID 1234.

ionice



kill is a **terminal** command line program with which processes can be stopped and finished.

The command is passed on according to the following pattern with a desired signal and the ID of the chosen process.

kill [OPTIONS] [-SIGNAL] PID

Common signals are:

TERM: Causes a process to end itself (Standard)

KILL: Forces the end of a process (through the system)

STOP: Stops a process

CONT: Allows a stopped process to continue

The following call sends a signal to the process 1234 that prompts it to end itself. Since no signal is given, *kill* sends the standard signal *TERM*.

kill 1234

Always give processes the chance to end themselves, and only force the action via *KILL* if the affected process doesn't react as intended.

kill -KILL 1234

If you only want to stop 1234 for a little while, use the following call to pause or restart the process:

kill -STOP 1234

kill -CONT 1234

Use the command *kill* with the option *-l* (*--list*) to show all possible signals that can be given to processes via *kill*.



Use `kill` in combination with a particular search term (/digitalguide/) to only end the processes whose names coincide (the first 15 characters are used to match).

killall

killall [OPTIONS] [-SIGNAL] [PROCESSNAME]

The option `-e` (`--exact`) allows you to extend the match to all characters of the process name.

Define process priorities

The command line directive `nice` indicates a process value between -20 and +19 at the start of a process in integer steps, after which the available computing power of the system is distributed. The range of -20 to +19 corresponds to the Linux priority levels 100 to 139. A process with a `nice` value of -20 has a higher priority than a process with a `nice` value of 19.

nice

The solo syntax reads:

nice [OPTION] [COMMAND]

Without additional specification, every process starts with a `nice` value of 0- Use the option `-n` to define the process priority. It should be noted that negative priorities can only be assigned with root permissions.

In the following example, the editor `nano` is started with a priority of 4:

nice -n 4 nano



Normally all of a user's dependent processes are automatically ended as soon as the terminal session is closed (i.e. via *exit*).

nohup

The command line directive *nohup* (short for *no hangup*) deletes a command from the current session and allows you to keep it running even when you log out of the system. The associated HUP signal (*hangup*) that normally causes a process to automatically terminate is suppressed to *nohup*.

The program call is based on the following pattern:

nohup COMMAND



The command `pgrep` and line program *pgrep* matches the list of running processes with a search term and outputs the respective PIDs if there are matches.

The general syntax reads:

pgrep [OPTIONS] Searchterm

By default, *pgrep* outputs the PIDs of all processes that contain the search term.

Example:

pgrep ssh

This will list all the processes that contain the search term *ssh* in the process name.

If the search is to be limited to only exact matches, then use the command along with the option *-x*.

Example:

pgrep -x sshd

This only lists the processes that are called exactly *sshd*.

If you would like to obtain the PID in addition to the process name, use *pgrep* with the option *-l*.

Similarly to *grep*, *pgrep* supports search terms based on regular expressions.



The `coreutils` and line program *pidof* outputs the process identification numbers (PIDs) of all of a program's processes.

Identify PIDs via *pidof* according to the following pattern:

pidof [OPTIONS] PROGRAM

With the following call, the IDs of all running processes of the program *nano* are output in the terminal.

pidof nano

If you would like to output only the first process ID, use *pidof* in combination with the option `-s` (short for *single shot*).

Stop and finish processes via search term

Like *kill*, the command *pkill* also sends a signal to a chosen process. The addressing isn't done by PID, though. Instead, a search term is given that matches the name of the running process. This can also be formulated as a regular expression.

pkill forwards the standard signal TERM, as long as no other signals are defined. The general syntax of the command reads:

pkill [OPTIONS] [-SIGNAL] [SEARCHTERM]

Additional options can be used to limit the command to the processes of a particular user (`-U UID`), the sub-processes of a particular parent process (`-P PID`), or the newest (`-n`) or oldest (`-o`) processes.

While *pkill* addresses all processes whose names contain the search term, the *killall* command only targets processes that are an exact match.

pidof

pkill



The command `ps` outputs a list of all running processes in the terminal.

ps

ps [OPTIONS]

If you need a detailed output, use *ps* with the options *-f* (detailed) or *-F* (very detailed).

For additional options, refer to your operating system's manual.

Obtain running processes as a tree structure

Use *pstree* to display all running processes in a tree structure.

pstree

The general syntax of the command reads:

pstree [OPTIONS]

The format and range of the output can be customized using various options.



The command and line directive *renice* allows you to customize the priority of a running process.

The general syntax reads:

renice PRIORITY [OPTIONS]

The addressing takes place with the help of options concerning the process ID (*-p PID*), group ID (*-g GID*), or a username (*-u USER*).

Example:

renice 12 -p 1234

The process with the ID 1234 is assigned the priority 12.

renice 3 -g 3456

All running processes of the group with the *GID 3456* are assigned a priority of 3.

sudo renice -6 -u peter24

All running processes of the user *peter24* are assigned a priority of -6.

If *renice* is used without options, the default value *-p* is assumed and the following string interpreted as a process ID.

renice



The `cor` and line directive *sleep* allows you to disrupt the current terminal session for a given time.

The general syntax of the command reads:

sleep NUMBER[SUFFIX]

If you use *sleep* without a suffix, the given number will be interpreted as time in seconds (s). You also have the option to disrupt the terminal session for minutes (m), hours (h), or days (d).

The following call disrupts the session for 4 minutes:

sleep 4m

The command is useful, for example, for delaying the execution of a subsequent command:

sleep 1h && reboot

The system waits one hour, and then afterward carries out the command *reboot*, which causes the system to restart.

sleep



The `taskset` command and line directive *taskset* is used for advanced process control, which is used in multiprocessor systems to assign processes or commands to specific processors.

The command requires root permissions and uses one of the following patterns:

taskset [OPTIONS] MASK COMMAND

taskset [OPTIONS] -p PID

Assigning a process or command to a processor happens using a hexadecimal bitmask. For example:

0x00000001 = Processor #0

0x00000003 = Processor #0 and #1

0xFFFFFFFF = All processors (#0 to #31)

Since assigning via bitmask like this isn't very intuitive, *taskset* is generally used with the option `-c (--cpu-list)` to enable a numerical assignment of processors (i.e. 0, 5 7, 9-11).

The following command tells the process 1234 to use processors 1 and 2:

taskset -p 1234 -c 1,2

taskset



The `cor` and `top` calls a dynamic overview of all running processes.

The call is based on the following pattern:

`top [OPTIONS]`

The output of the process information can be adjusted using various options. The `top` process overview (among others) supports the following hotkeys to sort through the outputs:

[P] = Sorts the output according to CPU load

[M] = Sorts the output according to storage requirements

[N] = Sorts the output numerically by PID

[A] = Sorts the output by age

[T] = Sorts the output by time

[U USERNAME or UID] = Filters the output by respective user

Use the hotkey [H] to display a help page, or [Q] to close the process overview.

top

Pager

Do you want to use your overview to keep track of multi-page file content? With a command line program from the pager category, you can select which sections are displayed in the terminal and scroll through the file in interactive mode if necessary.

Command	Description



The package `1&1` *head* is used to output the first part of a file.
(/digitalguide/)

The general syntax of the command reads:

head [OPTIONS] File

Use the option *-n NUMBER_LINES* to define how many lines are to be output, starting at the beginning.

Example:

head -n 3 example.txt

This will output the first three lines of the file *example.txt*. Without line specification, *head* outputs the first 10 lines of the given file.

Display text files in the terminal

The command line program *less* enables the display of the content of a text file in the terminal.

The general syntax reads:

less [OPTIONS] FILE

The output is automatically in interactive mode. This allows you to scroll through the selected document or search by keyword.

The [Q] key ends the interactive reading mode. Other control keys and available options can be found in the program's manual.

head

less



The package `more` fulfills the same function as `less`, but offers a smaller range of functions.

Use `more` according to the following pattern to call a text file and its content in the terminal:

`more [OPTIONS] FILE`

more

The command line program always displays a complete screen page of the chosen file. If a file contains multiple pages, `more` starts an interactive mode that allows you to scroll through the document using control keys or search by keyword.

The [Q] key ends the interactive mode. Other control keys as well as available options can be found in the manual of your operating system.

Output the last lines of a file

tail

While `head` displays the first 10 lines of a chosen file by default, `tail` outputs the last 10.

Both pages are used according to the same pattern (refer to `head`).

Editors

Under Linux, you don't need a graphical text editing program to customize configuration files, edit code snippets, or draft short notes. **Simple text editors** can be easily called up in the terminal without time delays. Here we present three programs that you should know.

Command	Description
---------	-------------



Emacs is a cross-platform text editor, which can be expanded as desired by a programming interface.

emacs

By default, Emacs starts with a graphical user interface, but can also be opened in the terminal using the option *-no-window-system*.

emacs -no-window-system

Emacs has an available integrated tutorial that you can call with the key combination [CTRL] + [H], [T].

The text editor Nano

Nano is a GNU reproduction of the terminal-based text editor Pico, used in the context of the mail client *Pine*. Nano offers a smaller range of functions than comparable editors (i.e. *Vim*), but is characterized by a particularly user-friendly handling.

The general syntax of the program call reads:

nano [OPTIONS] FILE

nano

The program opens the given file in an editing window in the terminal.

If you call Nano without file names, a new text file can be created that's stored in the currently selected directory.

nano [OPTIONS]

The key combinations for controlling the program are listed at the bottom of the editing window. Other information about Nano can be found in the program's manual.



Vim (short for *Vi Improved*) is a further development of the text editor *Vi* that stands out due to numerous extensions such as syntax highlighting, a comprehensive help system, native scripting, automatic code completion, and visual text selection.

The open source program offers various modes of operation for editing pure text files, and can be used either in the terminal or as a stand-alone application with a graphical user interface (GVim). A central application area of the program is the editing of program code.

If you start Vim in the console, the operation is done via keyboard. Generally, the program is called together with a text file according to the following pattern:

```
vim [OPTIONS] FILE
```

Opened files load Vim into a buffer. All changes that you make to the open file are also kept here. If you open Vim without specifying a file, the program starts with an empty buffer. The original file is not adapted until the memory operation is provided by the corresponding key combination. If no files exist that match the name given in the program call, one is newly created as part of the memory process.

Vim offers the program *vimtutor* as a comprehensive introduction, which is also started from the command line.

Our [basics article on the text editor Vim](#) ([server/tools/linux-editor-effective-code-editing-with-vim/](#)) also offers additional information on the installation and various operating modes of the program.



Network management is also managed easily from the terminal in Linux. Whether you want to test the connection, request DNS information, configure the interface, or transfer files to another computer in the network, with the following programs a single command is sufficient to put your project into motion.

Command	Description
arp	Display and manipulate the ARP cache
	The command line program <i>arp</i> allows you to access and manipulate the ARP cache of your operating system.
	Use <i>arp</i> without a modifier to output the content of the ARP table in the terminal.
	<i>arp</i>
	You can also use the option <i>-a</i> to limit the output to entries for a particular hostname (or an IP address).
	<i>arp -a HOSTNAME</i>
	Example:
	<i>arp -a example.com</i>
	If you want to create an ARP entry, use a program call with the option <i>-s</i> according to the following pattern:
	<i>arp -s HOSTNAME MAC_ADDRESS</i>
	Example:
	<i>arp -s example.com 00:05:23:73:e6:cf</i>
	If a particular entry is to be deleted, use <i>arp</i> with the option <i>-d</i> :
	<i>arp -d HOSTNAME</i>



`dig` is a lookup tool that can be used to request information from the DNS server and output it in the terminal.

The command line program is generally used according to the following syntax to request the IP address and other DNS information on a given domain name:

```
dig [ @SERVER ] [ DOMAIN ] [ TYPE ]
```

SERVER is the DNS server that is to be searched for the desired information. If no server is given, *dig* identifies the standard DNS server from the file */etc/resolv.conf*.

DOMAIN stands for the domain name from which the DNS information should be identified.

TYPE is used to specify the type of query, i.e. ANY (all entries), A (IPv4 record of a host), or AAAA (IPv6 record of a host). The standard request type is defined as A.

Use *dig* with the option *-x* to request the domain name for a given IP address as part of a reverse lookup.

```
dig [ @SERVER ] [ -x IPADDRESS ]
```

The arguments NAME, TYPE, and CLASS are not needed in this example.



With the **1&1** command line program *ftp*, most Linux distributions have a pre-installed client program for data transfer via FTP (File Transfer Protocol). This gives you the possibility to exchange files between the local system and another computer in the network.

Use *ftp* according to the following syntax to establish a connection to the FTP server of the target computer:

ftp [OPTIONS] [HOST[PORT]]

The addressing takes place via host name or IP address. Specifying a port number is optional.

In general, you're asked for a username and corresponding password when establishing a connection.

If the login is successful, *ftp* starts a command line interpreter that accepts user input in the form of commands. The program supports various commands for searching and managing the file system of the target computer, as well as transferring files from one system to another.

Obtain and configure network interfaces

The command line program *ip* is part of the program collection *iproute2*, with which network interfaces are requested and configured via the terminal.

The general syntax of the command reads:

ip [OPTIONS] OBJECT [COMMAND [ARGUMENT]]

Which action is carried out by *ip* is defined with the help of objects, subcommands, and their arguments.

The program supports various objects, such as *address* (IP address), *link* (network interface), *route* (entry in the routing table), or *tunnel*, to which subcommands

ftp

ip

such as *add*, *change*, *del*, *list*, or *show* can be added.

Digital Guide

For example, if you would like to access the IP address of a particular network interface (i.e. *eth0*), use the command *ip* in combination with the object *address*, the command *show*, and the argument *dev eth0*:

```
ip address show dev eth0
```

You can also give the objects and commands in shorthand:

```
ip a s dev eth0
```

If you would like to output all information to a network interface (i.e. *eth0*), use the command line directive *ip* with the object *link*, the command *show*, and the argument *dev eth0*:

```
ip link show dev eth0
```

or:

```
ip l s dev eth0
```

To activate or deactivate an interface like *eth0*, enter the following:

```
ip link set eth0 up
```

```
ip link set eth0 down
```

With its large range of functions, the program collection *iproute2* replaces a number of older network tools such as *ifconfig*, *route*, and *netstat*.

A list of all possible options, objects, subcommands, and arguments for the command line directive *ip* as well as information on other *iproute2* programs can be found in the manual of your operating system.

Obtain and configure WLAN interfaces

The command line program *iw* is used for the configuration of WLAN interfaces and is established as a current alternative to *iwconfig*.
(/digitalguide/)

The call is based on similar syntax to that of the *ip* command:

iw [OPTIONS] OBJECT [COMMAND]

Possible objects are:

dev NAME_OF_INTERFACE = Network interface

phy NAME_OF_DEVICE = WLAN device (by name)

phy#INDEX_OF_DEVICE = WLAN device (by index)

reg = Regulatory agent for the configuration of regional and country settings

Use *iw* with the command *help* to display the program syntax as well as possible options and commands.

iw help

The following is an application example of the command line program *iw*:

Output device settings of all WLAN interfaces:

iw list

Access connection status (transfer rate and signal strength) of a WLAN interface (i.e. *wlan0*):

iw dev wlan0 link

Scan WLAN environment:

iw dev wlan0 scan

Use *iw* in combination with the command *scan* to output all WLAN networks in the reception area as well as their properties (radio channel, encryption, signal strength, etc.).

Digital Guide Read out regional settings:

`iw reg get`
(/digitalguide/)

Modify regional settings:

`iw reg set US`

Request device properties (i.e. from `wlan0`):

`iw list dev wlan0`

Detailed device properties:

`iw dev wlan0 station dump`

Query events:

`iw event`

The options `-f`, `-t`, and `-r` deliver extended output with error messages regarding connection status and time.

Obtain network interface status

The command line program `netstat` ([server/tools/introduction-to-netstat/](#)) is used to query the status of network interfaces.

The general syntax of the command reads:

`netstat [OPTIONS]`

Use `netstat` without option to output all open sockets in the terminal.

You can also use the following options to see the routing table (`-r`), interface statistics (`-i`), masked connections (`-M`), or network link messages (`-N`).

An alternative to `netstat` is contained in the `iproute2` program collection program `ss`.

netstat



Like *dig*, *nslookup* (server/tools/nslookup/) is also a name resolution service. The command line program is available in two modes: interactive and non-interactive.

The interactive mode starts when the command *nslookup* is entered in the terminal without any additional information.

nslookup

The program now accepts commands. For example, type a hostname (domain) to get the corresponding IP address.

You can also start a reverse lookup request by entering an IP address and then outputting the associated hostname.

The program *nslookup* automatically uses the DNS server pre-installed in the system.

Enter the command *exit* to end *nslookup*.

If you want to use *nslookup* in non-interactive mode, call the program in combination with a hostname or IP address.

nslookup [OPTIONS] [HOST/IP]

Since the program is officially outdated, users are encouraged to rely on *dig* instead.

nslookup



Use the **1&1** command line program *ping* to test the accessibility of other computers in the network.

The command is based on the following syntax:

ping [OPTIONS] TARGET

To check the network connection, *ping* sends a small data package to the given target system (hostname or IP) and analyzes the time until the answer is received.

Together with the round-trip time (RTT) – the time span between sending the data package and receiving an answer – *ping* also writes the IP address of the target system in the terminal. So the command line program is also suitable for determining the IP address for a domain.

If *ping* is used without options, then the program runs until it's ended manually with the key combination [CTRL] + [C], and sends the target system a *ping* request every second.

If you want to define an end time when you call the program, use the options *-c NUMBER* (number of ping requests that will be sent) or *-w SECONDS* (time span in seconds, after which *ping* will end itself).

Display and edit the IP routing table

With the command line program *route*, the IP routing table of the core can be requested and edited.

The command is based on the following syntax:

route [OPTIONS]

route [OPTIONS][add/del] [-net/-host] TARGET

Use the command without options to display the complete routing table of the core:

ping

route



If you want to set a route to a network, use the subcommand *add*.

```
route add -net 10.0.0.0
```

If the target is a subnet, then the subnet mask has to be specified using the *netmask MASK* option:

```
route add -net 10.0.0.0 netmask 256.245.155.0
```

You can also set up a route to a computer:

```
route add -host 218.89.72.191
```

If the system is available over multiple network interfaces, then the option *dev INTERFACE* needs to be used to specify which interface should be used:

```
route add -net 10.0.0.0 netmask 256.245.155.0 dev eth0
```

If a target can only be reached via a router, this must also be specified with the *gw ROUTER* option.

```
route add -net 10.0.0.0 netmask 256.245.155.0 gw 10.0.1.261
```

If you want to delete a route, use the subcommand *del*.

```
route del -host 218.89.72.191
```



The command line program *rsync* enables you to synchronize files locally or over a network. For this purpose, the size and modification time of the concerned files are compared.

If the source and the target are located on the same system, then different files are copied completely. For synchronization over a network, *rsync* uses a delta transfer algorithm so that only altered file components have to be transferred from the source file carrier to the target system.

The syntax of the call reads:

rsync [OPTIONS] SOURCE(S) TARGET

Example:

rsync -a home/user/documents/ /home/user/backup

All files from *home/user/documents* are compared with the files in the directory */home/user/backup*.

The command *rsync* is generally performed with the option *-a*, which ensures that all subdirectories and symbolic links are copied and all user rights take effect.

Transfer files via SFTP

The command line program *sftp* functions like *ftp* to transfer data in the network. But here, all operations are performed via an encrypted SSH connection (secure shell).

Like *ftp*, *sftp* creates a connection to a target computer in the network and then starts an interactive command mode.

rsync

sftp



With **scp** (short for *secure copy*), another program for secure data transfer in the network is available directly via the terminal: *scp* copies data from one computer to another and uses the network protocol SSH.

The client program functions in the same way as the file option *cp*, but is used system-wide according to the following syntax:

```
scp [OPTIONS] FILE [[user@]remote_host:]PATH
```

When specifying the path of the remote computer, the username and the respective hostname are placed in front. Local files are explicitly addressed using relative or absolute paths.

Example:

```
scp/home/max/images/image.jpg  
max@example.com:/home/max/archive
```

The file *image.jpg* is copied from the local *images* directory to the *archive* directory on the target computer with the address *example.com*.

The program *scp* also supports data transfer in the opposite direction as well as between two remote systems.

```
scp [OPTIONS] [[user@]host:]FILE PATH
```

```
scp [OPTIONS] [[user@]host1:]FILE  
[[user@]host2:]PATH
```

Additional options allow you to make adjustments to the transfer mode and the encryption settings.



Use the **1&1** command line directive *traceroute* according to the following pattern to track the transport path of an IP data package between your system and a destination computer.

traceroute

traceroute [OPTIONS] HOSTNAME

Via *traceroute* you can identify which router and internet nodes an IP package passes on its way to the target computer – for example, to investigate the cause of a delay.

Output terminal names

tty

The command line directive *tty* outputs the file names of the terminal that are defined as the standard input.

The general syntax of the command reads:

tty [OPTIONS]

Archive and compress

Linux offers various technologies with which files can be packed and compressed in archives. It should be noted that not every **archive** contains a **compression**. So *tar* – a program for the archiving of files – is usually combined with a compression program like *gzip*, *bzip2*, or *xz*.


Command

Description

tar

Write and extract files in the tar archive

The command *tar* stands for *tape archiver*, a program that was originally developed to secure data on tape drives. Even today, *tar* is one of the most popular programs for archiving data under Linux.

The program allows you to write various files and directories sequentially into a *tar* file and use it as a backup  recovery if needed. Unlike the zip format common in Windows, all user rights of the archived file are retained even after unpacking.

The command line program *tar* is called according to the following syntax:

```
tar [OPTIONS] FILES
```

If you want to create a new archive, use *tar* with the options *-c* (create new archive) and *-f* (write archive to a given file or read from it).

In the following example, the files *file1.txt* and *file2.txt* are written into the newly created archive *example.tar*.

```
tar -cf example.tar file1.txt file2.txt
```

If you want to see the content of an archive, use *tar* with the options *-t* (display archive content), *-v* (detailed output), and *-f* (see above).

```
tar -tvf example.tar
```

If archived files are to be unzipped into the current folder, then use *tar* with the options *-x* (extract files from archive) and *-f* (see above).

```
tar -xf example.tar
```

tar offers more possibilities with *-j* (bzip2), *-J* (xz), *-z* (gzip), and *-Z* (compress) that allow you to compress or decompress archives when you call another program during the packing and unpacking processes.

In the following example: the files *file1.txt* and *file2.txt* are archived in *example.tar.gz* and compressed with *gzip*.

```
tar -czf example.tar.gz file_1.txt file_2.txt
```



(/digitalguide/tar -xzf *example.tar.gz*

Compress or decompress files with gzip

gzip (short for *GNU zip*) is a program with which you can easily compress or decompress files via the command line.

The general syntax of the command reads:

gzip [OPTIONS] FILE(S)

For example, use *gzip* according to the following pattern to transfer the file *example.txt* to the compressed format *example.txt.gz*:

gzip example.txt

Note that by default, *gzip* deletes the original file as part of the packing process. Prevent this by using the option *-k*.

gzip -k example.txt

The program can be used for multiple files at the same time, if necessary. Each output file is converted into a separate *gz* file.

So the command...:

gzip example_1.txt example_2.txt example_3.txt

...generates the files *example_1.txt.gz*, *example_2.txt.gz*, and *example_3.txt.gz*.



A popular alternative to *gzip* is the command line program *bzip2*. This uses the same syntax as *gzip*, but is based on a three-stage compression process which allows for a significantly higher compression ratio.

First, the given files are wrapped block-wise by the Burrows-Wheeler transformation and then by the move-to-front transformation. The actual data compression finally happens with a Huffman coding.

Files that are compressed with *bzip2* use the file ending *-bz2*. Use *bzip* according to the following pattern to compress files:

bzip2 [OPTIONS] FILE(S)

bzip2 can also be applied to *tar* archives.

The decompression is analog with *gzip* and runs with the help of option *-d*. The command *bunzip2* is also available.

Users pay for the high compression ratio with a comparatively long runtime.

bzip2 / bunzip2



The command line program `xz` converts files in the same-named data compression format `xz`. The program call uses the same pattern as *gzip* and *bzip2*.

`xz [OPTIONS] FILE(S)`

Files that are compressed with `xz` use the file ending `.xz`. The decompression functions as with *gzip* and *bzip* with the option `-d`. The command *unxz* can also be used.

Like `gz` and `bz2` files, `xz` files are also not archive files. If you would like to write multiple files into the same compressed `xz` file, you also have to rely on the archiving tool *tar* with this compression program.

`xz` supports various compression algorithms. The Lempel-Ziv-Markov algorithm (LZMA/LZMA2) is used by default.

Write and extract files to archive file

The archiving program *cpio* (short for *copy in, copy out*) allows you to write data in an archive file (`.cpio`) and extract data from it.

xz

cpio

A detailed description of the command line programs listed here can be found in our basics article on the topic "[Archiving and Compressing Using Linux \(server/tools/archiving-and-compression-using-linux/\)](#)". Additional information on compression methods, as well as the definition of deduplication, can be found in our [article on data reduction \(server/know-how/data-reduction-through-deduplication-and-compression/\)](#).

Partition management

If you want to access a file system on another partition in Linux, you first have to integrate it into the directory structure of your operating system. This is called "**mounting**" a partition. If necessary, this can happen via the graphical user interface. Command line programs like *lsblk*, *blkid*, and *mount* also offer the ability to request information about connected block storage devices and to mount or unmount them when necessary.



mount /unmount

Integrate file systems
(/digitalguide/)

If a file system is to be integrated in the directory structure of the operating system via the directory structure, then the command line program *mount* is used on Linux.

The general syntax of the command reads:

mount [OPTIONS] DEVICE MOUNTPOINT

DEVICE = Path to the device file of the storage device that you want to mount as the partition.

MOUNTPOINT = The location in the directory structure of your operating system where you want to mount the partition. The mountpoint is usually specified as an absolute path.

Example:

mount /dev/sdd /media/usb

The device *sdd* is mounted in the directory */media/usb*.

In general, Linux automatically recognizes the respective file system of the device. If this isn't the case, then the option *-t* gives you the option to explicitly share the file system (i.e. *ext4*):

mount -t ext4 /dev/sdd /media/usb

If a previously integrated file system is to be unmounted, then use the command *umount*:

umount [OPTIONS] DEVICE

or

umount [OPTIONS] MOUNTPOINT

If you want to output all file systems that are integrated in your operating system, use the command *mount* with the option *-l*.

Digital Guide (/digitalguide/)

mount -l

The output can be limited to file systems of a particular type via *-t*.

List information on connected block storage devices

Use the command *lsblk* (short for *list block devices*) to represent all connected block storage devices and partitions as a tree structure. These don't necessarily have to be involved.

The call is based on the following syntax:

lsblk [OPTIONS]

The output includes the following information:

NAME = Device name (i.e. sda) or partition name (i.e. sda1, sda2, etc.)

MAJ:MIN = major:minor Device number

RM = Exchange medium (1 = applicable, 0 = not applicable)

SIZE = Device storage size

RO = Read-only device (1 = applicable, 0 = not applicable)

TYPE = Device type

MOUNTPOINT = Mounting point

If necessary, the output and a list of desired attributes can be individually modified using the *-o (--output)* option to retrieve additional information, like the identification number (UUID), file system (FSTYPE), or the state (STATE).

lsblk



```
lsblk -o NAME,FSTYPE,UUID,  
1&1  
(/digitalguide/$SIZE,OWNER,GROUP,MODE,TYPE,MOUNTPOINT
```

In the standard settings, empty storage devices are bypassed. If you also want to include these in the overview, use *lsblk* in combination with the option *-a* (*--all*).

If you only want to request information on a particular device, use *lsblk* according to the following pattern:

```
lsblk [OPTIONS] DEVICE
```

Example:

```
lsblk /dev/sda
```

List information on connected block storage devices

Similar to *lsblk*, *blkid* also outputs information on connected block storage devices.

Use *blkid* according to the following pattern to obtain the identification number (*UUID*) and file system type (*TYPE*) of all connected block storage devices.

```
blkid [OPTIONS]
```

For tabular output, use the *-o* option in combination with the value *list*.

```
blkid -o list
```

You can also limit *blkid* to a chosen device:

```
blkid [OPTIONS] DEVICE
```

Example:

```
blkid /dev/sda1
```



The command and line program *dd* enables a copying process in which data is read out bit for bit from an input file (*if*) and written into an output file (*of*).

The program call is based on the following syntax:

```
dd if=Source of=Target [OPTIONS]
```

As the source and target, you can specify individual files as well as entire partitions (i.e. */dev/sda1*) or a complete storage device (i.e. */dev/sda*).

Example:

```
dd if=/dev/sda5 of=/dev/sdb1
```

The complete fifth partition of */dev/sda* is copied bit-exactly from the first partition of */dev/sdb*.

The copying process can be limited to any number of memory blocks of the desired size via options.

dd

Miscellaneous

The following list contains additional Linux basic commands that don't belong to any of the previous categories.

Command	Description



The `cor` and line program *alias* enables you to define nicknames for program calls.

Use *alias* according to the following pattern:

```
alias NICKNAME= 'COMMAND'
```

Replace the placeholder `COMMAND` with any command line directive, including options. This will link the inserted string for the placeholder `NICKNAME`.

Example:

```
alias ll='ls -l'
```

The string `ll` is defined as the alias for the command `ls` with the option `-l` (detailed output).

Run time-controlled command

Call the command line program *at* according to the following pattern to run a time-controlled command.

```
at TIME
```

Example:

```
at 10:00 AM 6/22/2017
```

Then enter the command and close the interactive mode with `[CTRL] + [D]`.

alias

at



Use `cal` according to the following pattern to output a calendar in the terminal.

`cal`

cal [OPTIONS] [[MONTH] Year]

Example:

cal 12 2017

The system outputs a month overview for December 2017.

Output string to the standard specification

`echo`

Use the command line directive *echo* to output strings line-by-line on the standard output (usually the terminal).

The general syntax of the command reads:

echo [OPTIONS] STRING

Prepare text files for printing

`pr`

Use the command line program *pr* to prepare text files for printing.

The general syntax of the command reads:

pr [OPTIONS] File

In the standard settings, *pr* generates a page header that contains the file name, current date, and page number.



The command `script` and line program *script* allows you to record a terminal session in the file *typescript*. If there's already a recording of a previous session in *typescript*, then it's overwritten.

The recording automatically starts with the program call:

script

Use the key combination [CTRL] + [D] to end the recording.

If you would like to save the recording in another file instead of in *typescript*, call *script* in combination with a file name or path.

script FILE

Output numerical series

Use the command *seq* to output a numerical series in the standard output. Define a start value, an end value, and an increment (optional).

seq [OPTIONS] STARTVALUE INCREMENT ENDVALUE

Example:

seq 0 2 100

The program counts from the start value 2 to the end value of 100 in increments of 2.

script

seq



The command line program *tasksel* serves as installation help for standard applications (mail server, DNS server, OpenSSH server, LAMP server, etc.). Use the tool to automatically install all packages and programs required for a task in the correct order.

Call *tasksel* with the option *--list-tasks* to output a list of all available standard applications.

tasksel --list-tasks

If you want to access more information about a standard application on the list, use *tasksel* with the option *--task-desc* and the corresponding task.

Example:

tasksel --task-desc mail-server

This outputs information about the “mail-server” task.

If you want to list all packages that belong to the “mail-server” task, use *tasksel* in combination with the option *--task-packages*.

tasksel --task-packages mail-server

To install all packages of a standard application, use the subcommand *install*. This requires root permissions.

Example:

tasksel install mail-server

The command line program initiates the installation of all packages that are necessary for the “mail-server” task.

tasksel



The command `&&` and line program *tee* is used to double the output of a program. One output is passed to the standard output, and another is written to the file given with the *tee* command.

The general syntax of the command reads:

tee [OPTIONS] FILE

tee is usually used in combination with the redirection operator Pipe (`|`).

ls |tee example.txt

The command *ls* lists the content of the current directory. The program output is delivered to the command line program *tee* via Pipe, which displays this in the terminal as well as in the file *example.txt*.

Measure the runtime of programs

Use the command *time* according to the following pattern to identify the runtime of programs that you've started over the terminal.

time [OPTIONS] Command [ARGUMENTS]

tee

time



Use *tr* to delete a desired character set or replace it with another. To do this, *tr* reads the data stream of the standard input and writes it to the standard output according to the desired modification.

If a character set is to be replaced by another, then *tr* is used with two arguments.

tr *OPTION CHARACTERSET1 CHARACTERSET2*

The second argument (CHARACTERSET2) replaces the first (CHARACTERSET1).

If you want to delete a character sequence, use *tr* with the option *-d* and enter the set to be deleted as the argument.

tr -d CHARACTERSET

The command line program is usually used in combination with redirection operators (< and >) to make modifications to files.

tr 'a-z' 'A-Z' < example1.txt > example2.txt

tr reads out the content of the *example1.txt* file, replaces the lower-case letters a through z with upper-case letters, and writes the output in the *example2.txt* file.



The core and line program *wall* allows you to send a message to all users registered on a system.

To send a communication, start the program with the following call:

```
wall
```

Confirm the program call with [Enter] and enter your message. Then confirm again with [Enter] and send with the key combination [CTRL]+[D].

All users registered on the system receive your message as a broadcast in the terminal.

Note: To be able to receive communications, you have to provide other users with a write access to your terminal. For this, use the command *mesg*:

```
mesg [y/n]
```

Obtain current status:

```
mesg
```

Provide write access:

```
mesg y
```

Deny write access:

```
mesg n
```

If you want to send file content to all registered users, use *wall* in combination with an input redirection and the respective file name:

```
wall < FILENAME
```



The core **1&1** and line program *watch* allows you to set a command to run at regular intervals.

watch

The program call is based on the following syntax:

```
watch [OPTIONS] COMMAND
```

The time interval at which the command given in *watch* will be run is defined with the option *-n SECONDS*.

End *watch* with the key combination [CTRL] + [C].

In the following example, the system is instructed to output the workload of the internal memory at 10-second intervals.

```
watch -n 10 free
```

Count lines, words, letters, characters, and/or bytes of a text file

The command line program *wc* short for (*word count*) outputs the number of lines, words, letters, characters, and/or bytes of a text file, by request.

The overall syntax of the command reads:

```
wc [OPTIONS] FILE
```

Example:

```
wc example.txt
```

Output:

```
14 18 143 example.txt
```

If *wc* is called without options, the output corresponds to the *LINES WORDS CHARACTERS FILE*. For a filtered output, the command line program supports the options: *-l* (lines), *-c* (bytes), *-m* (characters), *-L* (length of the longest line), and *-w* (words).

wc



The `&&` command line program *xargs* allows you to transfer the output of a previous command to a new command as an argument. Generally, this is used with the Pipe (`|`) as a diversion operator.

Use *xargs* according to the following syntax:

```
COMMAND1 | xargs [OPTIONS] COMMAND2
```

xargs can be used in combination with the command *find*, for example.

In the following example, *find* identifies all files in the current directory that fit the search term **.tmp*, and outputs their names to the standard output. There, the file names of *xargs* are accepted and passed as arguments to the command *rm*.

```
$ find . -name '*.tmp' | xargs rm
```

xargs

The overview presented here doesn't claim to be complete, but includes basic Linux commands with selected application examples for everyday work with unix-like operating systems. A comprehensive description of the command line programs presented here, as well as all other commands, can be found in the manual of your operating system. An online version of these help-and-documentation pages are available via the Linux man-pages project from Michael Kerrisk on [kernel.org/doc/man-pages](https://www.kernel.org/doc/man-pages/) (<https://www.kernel.org/doc/man-pages/>).

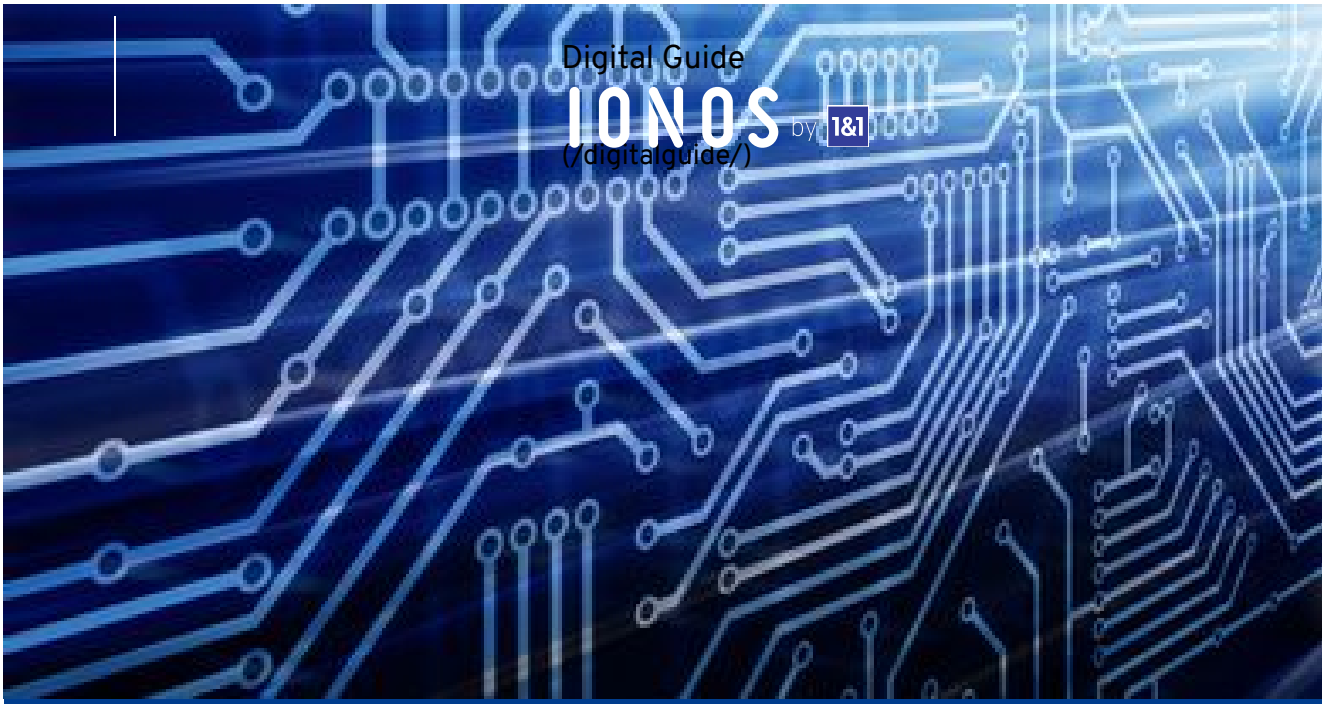
© 08.05.20 | Configuration (server/configuration/)

How did you like the article? 5 ★★★★★ (2)

tags overview / m/c

[Tutorials \(tags/tutorials/\)](#) [Tools \(tags/tools/\)](#) [Linux \(tags/linux/\)](#) [Encyclopedia \(tags/encyclopedia/\)](#)

Related articles



(server/configuration/provide-raspberry-pi-with-a-static-ip-address/)

Equip Raspberry Pi with a static IP address (server/configuration/provide-raspberry-pi-with-a-static-ip-address/)

🕒 29.08.2019 | Configuration (server/configuration/)

Both the private Raspberry Pi IP address and the public IP address of the internet connection are constantly changing for most users. Such dynamic IP addresses aren't practical for the use of a minicomputer. But if you want to use server software effectively on your Raspberry Pi, you need a static IP address for the computer in the local network. The internet access used to make the server...



Related products




Domain Names

See packages ▶

Web hosting for agencies

Provide powerful and reliable service to your clients with a web hosting package from IONOS.
Digital Guide



View packages 
(/digitalguide/)

Popular Articles

[.ly/](#) [.ly/I](#) [ly/](#) [os.I](#) [-tee](#)
[Iid](#)

About 1&1 (<https://www.ionos.com/about>)

Terms and Conditions (<https://www.ionos.com/terms-gtc/general-terms-and-conditions/>)

Privacy Policy (<https://www.ionos.com/terms-gtc/terms-privacy>)

Help Center (<https://www.ionos.com/help>) Tell a friend (<https://www.ionos.com/referral>)

© 2020 1&1 IONOS Inc. (<https://www.ionos.com>)