

Introduction to Linux system

You should be able to navigate a system on a command line level. Especially in HPC you should not count on the existence of GUIs. Even though we will do all exercises in Jupyter Hub you still need to navigate the command line for setting up everything. And it does no harm to know how to use the command line ;)

Side note: Windows does offer its own command line (CMD, Powershell), but we will not cover this. If you want to stay with Windows and work locally you can give WSL a try.

There are different kinds of commands you can use within your terminal: shell builtins and programs. But for now we can ignore this distinction. It's nice to know that the distinction exists but you do not need to worry about it. And: **Be careful about case sensitivity.**

Tip: If you want to cancel the execution of any command, just press `ctrl + c` to abort it or `ctrl + d` to terminate your current shell.

Basics

The basic commands allow the general navigation within the command line. We will discuss the `man` command in detail and will spare details with the following commands. You can look them up on their own man page.

`man` pages

You want to know what a command does? Which flags are available? Which similar/associated tools exist? Look it up in the `man(ual)` page! It has nearly everything you need to know.

Most commands have a man page, even the `man` command itself. Executing `man man` gives you a short manual on how to use `man`.

For example, you want to know, what `top` does? Just look it up in the man page: `man top`.

The structure of man pages is always similar. Usually it starts with three sections: 1. NAME 2. SYNOPSIS 3. DESCRIPTION

The synopsis tells you, how you call the command (which options are required/optional, which flags are available).

Flags allow to modify how a command is executed, each command may have its own set. Very often there are two kinds of flags: 1. abbreviated (e.g. `ls -a`) 2. written out (e.g. `ls --all`)

It does not matter, which you use. Writing flags out costs more time but helps to remember, how the execution is modified. Flags can be combined, if you use

```

[001]                                     Manual page built                                     [001]
NAME
    man - an interface to the system reference manuals

SYNOPSIS
    man [man options] [[section] page ...] ...
    man -k [keyword options] keyword ...
    man -k [man options] [section] keyword ...
    man -f [whatis options] keyword ...
    man -l [man options] file ...
    man -w [-w] [man options] page ...

DESCRIPTION
    man is the system's manual pager. Each page argument given to man is merely the name of a program, utility or function. The manual page associated with each of these arguments is then found and displayed. A section, if provided, will direct man to look only in that section of the manual. The default action is to search in all of the available sections following a pre-defined order (see DEFAULTS), and to show only the first page found, even if page exists in several sections.

    The table below shows the section numbers of the manual followed by the types of pages they contain.

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

    A manual page consists of several sections.
    Conventional section names include NAME, SYNOPSIS, CONFIGURATION, DESCRIPTION, OPTIONS, EXIT STATUS, RETURN VALUE, ERRORS, ENVIRONMENT, FILES, VERSIONS, CONFORMING TO, NOTES, BUGS, EXAMPLE, AUTHORS, and SEE ALSO.

    The following conventions apply to the SYNOPSIS section and can be used as a guide in other sections.

    hold text           type exactly as shown.
    italic text         replace with appropriate argument.
    [-whc]             any or all arguments within [ ] are optional.
    -w|-b              options delimited by | cannot be used together.
    argument ...       Argument is repeatable.
    [expression] ...    entire expression within [ ] is repeatable.

    Exact rendering may vary depending on the output device. For instance, man will usually not be able to render italics when running in a terminal, and will typically use underlined or coloured text instead.

    The command or function illustration is a pattern that should match all possible invocations. In some cases it is advisable to illustrate several exclusive invocations as is shown in the SYNOPSIS section of this manual page.

EXAMPLES
    man ls
    Display the manual page for the ls (program) ls.

    man man-2
    Display the manual page for macro package man from section 2. (This is an alternative spelling of "man 2 man".)

    man(2)
    Display the manual page for macro package man from section 2. (This is another alternative spelling of "man 2 man". It may be more convenient when copying and pasting cross-references to manual
Manual page built line 1 press h for help or q to quit

```

Figure 1: Output of `man man`

the abbreviated version, they can even be merged together. Instead of writing `mkdir --parents --verbose foo` you could also write `mkdir -pv foo`.

If you want to know, what the effect of a specific flag is, you can look it up in the description section. Since those can be rather large, you can also search for a flag (which might feel unusual at first). To search for something, press `/`, type in the expression you want to search for and hit `ENTER`. You now can see the first found occurrence. Pressing `n` or `N` allows you to jump to the next/prior occurrence. To leave the man pages, press `q`.

Note: Shell builtins do not have their own man page. Instead you are redirected to the man page of `BASH_BUILTINS`, where all builtins are collected and explained.

pwd

Print the path of your current working directory.

Interesting flags: `-P`: Resolve symlinks

Example:

```
$ pwd
/home/username
```

ls

List the content of a directory. If a path is omitted, the content of the current working directory is shown; files are sorted alphabetically by default.

Interesting flags: - **-l**: list files and show additional information like permissions, owner, file size (bytes) and modification time. - **-h**: Eases reading the file size (e.g. if using **-l**) by using suffixes like 1K, 234M or 2G. - **-a**: Show hidden files (prefixed with a dot) - **-R**: List subdirectories recursively - **-r**: Reverse order of listed files - **-t**: Sort by time

Example:

```
$ ls
Documents Downloads tmp

$ ls -a
Documents Downloads tmp .venv
```

```
$ ls -l Downloads
-rw-r--r--.  1 user user  48369718 Jul  6  2020 cats.mp4
drwxr-xr-x.  2 user user    4096 Jul  6  2020 videos
```

Wildcards

Wildcards are handy to match several files without having to write them out all together. A wildcard is a placeholder for no/one/several characters. They can be combined in complex patterns, have a look at this link.

The most important ones are the following: - **?**: Any single character - *****: any number of characters (including zero) - ****: Escape characters, which could be interpreted as wildcard

Example:

```
$ ls
aa.txt  ab.csv  bc.txt

$ ls *b*
ab.csv  bc.txt

$ ls ?b*
ab.csv
```

tree

Display contents of directories like a tree

Interesting flags: - **-d**: List only directories - **-L level**: Max display depth

Example:

```
$ tree -L 1 /
/
  bin -> usr/bin
```

```
boot
dev
etc
home
lib -> usr/lib
lib64 -> usr/lib64
lost+found
media
mnt
opt
proc
root
run
sbin -> usr/sbin
srv
sys
tmp
usr
var
```

mkdir

Create a new directory.

Interesting flags: - -p: Parent directories are created if they do not exist, yet

Example:

```
$ ls -a
. ..
```

```
$ mkdir foo
```

```
$ ls
foo
```

```
$ mkdir bar/baz
```

```
mkdir: cannot create directory 'bar/baz': No such file or directory
```

```
$ mkdir -p bar/baz
```

```
$ tree
```

```
.
├── bar
│   └── baz
└── foo
```

cd

Change the directory to **dir**. If **dir** is omitted, the value of **\$HOME** is used (which points usually to the user's home directory). You can use absolute or relative directory paths. There are two special directories: **.** (current directory) and **..** (parent directory):

Example:

```
$ pwd
/home/user
```

```
$ ls
bar foo
```

```
$ cd bar
```

```
$ pwd
/home/user/bar
```

```
$ cd ..
```

```
$ pwd
/home/user
```

```
$ cd bar/..
```

```
$ pwd
/home/user
```

```
$ cd bar/./.
```

```
$ pwd
/home/user/bar
```

```
$ cd
```

```
$ pwd
/home/user
```

mv

Rename or move a file

Example:

```
$ ls -l
drwxr-xr-x. 4 user user 4.0K Oct 22 11:54 bar
```

```
drwxr-xr-x. 4 user user 4.0K Oct 22 11:54 foo
```

```
$ mv foo bar/foo2
```

```
$ tree
```

```
.
├── bar
│   ├── baz
│   └── foo2
```

cp

Copy a file

Interesting flags: - -r: Copy directory recursively

Example:

```
$ cp bar bar2
```

```
cp: -r not specified; omitting directory 'bar'
```

```
$ cp -r bar bar2
```

```
$ tree
```

```
.
├── bar
│   ├── baz
│   └── foo2
└── bar2
    ├── baz
    └── foo2
```

rm

Delete files. Pay attention, what you delete! Using **rm** immediately deletes files, **there is no trash bin!** Especially using wildcards like ***** can be dangerous if you do not pay close attention! You could test your command call by replacing **rm** with **ls** first to see, which files would be deleted.

Deleting systemically relevant directories or files could damage your system irreversibly.

Interesting flags: - -f: Force removal, missing files are ignored - -i: Ask before every removal of a file - -r: Remove directory and its contents recursively (a directory cannot be removed if neither **-r** nor **-d** is used)

Example:

```
$ tree
```

```
.
```

```

bar
  baz
  foo2
bar2
  baz
  foo2

$ rm bar
rm: cannot remove 'bar': Is a directory

$ rm -r bar

$ tree
.
├── bar2
│   ├── baz
│   └── foo2

```

Piping

The shell offers some comfort features: redirection and piping. We will not cover redirection but give a short introduction into piping.

To use the output of a program as input of a second program, the output of the first program can be piped into the second program. To do this, you just need to connect both programs with the pipe (`|`) operator (on german keyboard layout between your `shift` key and `y` key).

To scroll through a large output, you could pipe it to `less`. Then you can scroll through the output with your arrow keys. Press `q` to exit this view.

If you are just interested into the first/last 10 lines you can pipe it to `head/tail` (providing `-n NUMBER` sets the number of lines to be shown).

Example:

```

$ ls -lR / | head
ls: cannot open directory '/boot/efi':
total 64
lrwxrwxrwx.  1 root root    7 Jan 26  2021 bin -> usr/bin
dr-xr-xr-x.  7 root root 4096 Oct 23 10:02 boot
: Permission denieddrwxr-xr-x. 24 root root 4300 Oct 25 08:31 dev

drwxr-xr-x. 204 root root 12288 Oct 25 08:32 etc
drwxr-xr-x.  4 root root  4096 Jan 26  2021 home
lrwxrwxrwx.  1 root root    7 Jan 26  2021 lib -> usr/lib
lrwxrwxrwx.  1 root root    9 Jan 26  2021 lib64 -> usr/lib64
drwx-----.  2 root root 16384 Apr 26  2019 lost+found

```

```
$ ls -lR /usr/bin |tail -n 3
-rwxr-xr-x. 1 root root 28672 Jan 26 2021 tetgen_to_cgns
-rwxr-xr-x. 1 root root 78496 Jan 26 2021 update_ngon
-rwxr-xr-x. 1 root root 45600 Jan 26 2021 vgrid_to_cgns
```

Additional material regarding piping and redirection is available [here](#)

history

Your input on the command line are stored within a (size-limited) history file. To print it to your terminal, just execute **history**. To ease handling the whole history output, you can pipe it to **less**. This command becomes handy, if you know, you already did something but forgot how you did it. If it is still part of your shell history, you can then search for it.

cat

cat is used to print the content of a file to the command line.

Example:

```
$ echo "Hello World" > foo.txt
```

```
$ cat foo.txt
Hello World
```

chmod

To change the permissions of any file (read, write, execute) you can use **chmod**. Since this is an extensive topic, we just reference material.

Tip: To enter a directory, it needs to have its execute permission set.

Environment variables

Environment variables are names with an associated stored value. They can be used by applications, which are executed in the same (sub-)shell and are very handy for scripting. The name is case-sensitive, it's good practice to use upper case names only.

The content of a environment variable can be accessed by prepending a dollar sign **\$**. To print it to the shell you can use **echo**, to unset it again you can use **unset**.

There are three ways to set an environment variable: - **KEY=value** - **KEY="Some other value"** - **KEY=value1:value2**

Examples:


```
$ echo $FOO
```

```
$ FOO=test
```

```
$echo $FOO  
test
```

```
$ unset FOO
```

```
$echo $FOO
```

If the value shall be accessible within a subshell, the environment variable needs to be exported first. For example this is needed, if it shall be used in a shell script, which is executed from the shell. Without exporting the variable you can achieve the same by prepending setting the environment variable on the same line like the script call.

Examples:

```
$ cat test.sh  
echo Hello $FOO
```

```
$ bash test.sh  
Hello
```

```
$ FOO=world bash test.sh  
Hello world
```

```
$ export FOO
```

```
$ bash test.sh  
Hello world
```

To get an overview of all exported environment variables, which are currently set within you shell, you can use `env`.

There are some environment variables, which are already set. Be cautious to not unset those, as this could mess with your work environment.

- **\$PATH**: The shell looks for executables within those path's. If an executable does not reside within such a path, it can only be executed by using its complete absolute path
- **\$USER**: Your username
- **\$HOME**: Absolute path of your home directory
- **\$PWD**: Absolute path to your working directory. Can be used to list files with their absolute path

```
ls -d $PWD/bar2/*  
/home/user/tmp/test/bar2/baz  /home/user/tmp/test/bar2/foo2
```

SSH

`ssh` is a rather complex topic, therefore we refer to DKRZ material, where also the public key procedure for Levante is described.

In the most simple case, you just execute `ssh` with your username, which is attached with an `@` symbol to the target hostname or -IP.

Example:

```
$ ssh user@levante.dkrz.de
user@levante.dkrz.de's password:
Permission denied, please try again.
user@levante.dkrz.de's password:
[user@levante0 ~]$
```

Based on `ssh` there is the possibility to copy data from/to any host from/to you local machine via `scp`. The syntax is simply `scp source target`.

Interesting flags: - `-r`: Recursively copy entire directory - `-o`: Allows to pass options to the `ssh` backend

Example:

```
$ scp local_file.txt user@levante.dkrz.de:
local_file.txt                                100% 195    45.9KB/s   00:00
```

`exit`

Type `exit` or press `ctrl + d` to close your current (sub-)shell.

More docu

- Basic Unix introduction
- Bash Scripting Guide