Introduction to the Linux Commandline

Release 1.4

Holger Dinkel, Frank Thommen & Thomas Zichner

February 14, 2014

Contents

Chapter 1

Introduction to the Linux Commandline

1.1 Why Use the Commandline

- It's **fast**. Productivity is a word that gets tossed around a lot by so-called power users, but the command line can really streamline your computer use, assuming you learn to use it right.

- It's **easier to get help**. The command line may not be the easiest thing to use, but it makes life a whole lot easier for people trying to help you and for yourself when looking for help, especially over the internet. Many times it's as simple as the helper posting a few commands and some instructions and the recipient copying and pasting those commands. Anyone who has spent hours listening to someone from tech support say something like, "OK, now click this, then this, then select this menu command" knows how frustrating the GUI alternative can be.

- It's nearly **universal**. There are hundreds of Linux distros out there, each with a slightly different graphical environment. Thankfully, the various distros do have one common element: the command line. There are distro-specific commands, but the bulk of commands will work on any Linux system.

- It's **powerful**. The companies behind those other operating systems try their best to stop a user from accidentally screwing up their computer. Doing this involves hiding a lot of the components and tools that could harm a computer away from novices. Linux is more of an open book, which is due in part to its prominent use of the command line.

- Many 'modern' bioinformatics tools (samtools, bamtools, ...) are written for the commandline in order to be run on clusters and to be incorporated in scripts.

1.2 General Remarks Regarding Using UNIX/Linux Systems

- **Test before run**. Anything written here has to be taken with a grain of salt. On another system - be it a different Linux distribution or another UNIXoid operating system - you might find the same command but without the support of some of the options tought

here. It is even possible, that the same option has a different meaning on another system. With this in mind always make sure to test your commands (specially the "dangerous" ones which remove or modify files) when switching from one system to the other.

- **The Linux/UNIX environment**. The behaviour of many commands is influenced or controlled by the so-called "environment". This environment is the sum of all your environment variables. Some of these environment variables will be shown towards the end of this course.

- **UPPERCASE, lowercase**. Don't forget that everything is case-sensitive.

- **The Filesystem**. Linux filesystems start on top at the root directory (sic!) "/" which hierarchically broadens towards the ground. The separator between directories or directories and files in Linux is the slash ("/").
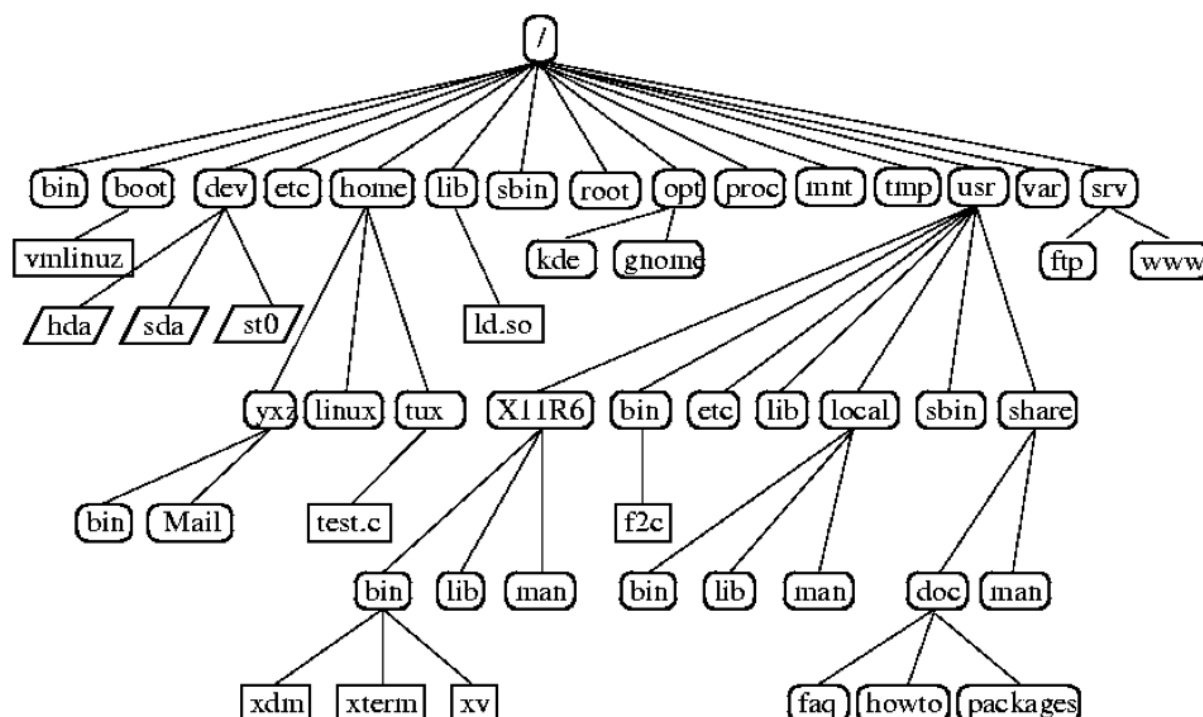
Figure 1.1: Depending on the Linux distribution you might or might not find all of above directories. Most important directories for you are `/bin` and `/usr/bin` (sometimes also `/usr/local/bin`) which contain the user software, `/home` which usually contains the users' homedirectories and `/tmp` which can be used to store temporary data (beware: Its content is regularly removed!).

Note: The terms "directory" and "folder" are used interchangeably in this document.

1.3   General Structure of Linux Commands

Many linux commands have options and accept arguments. Options are a set of switch-like parameters while arguments are usually free text input (such as a filename).

For example, in the commandline `ls -l /usr/bin`, `ls` is the command, `-l` is an option and `/usr/bin` qualifies as an argument.
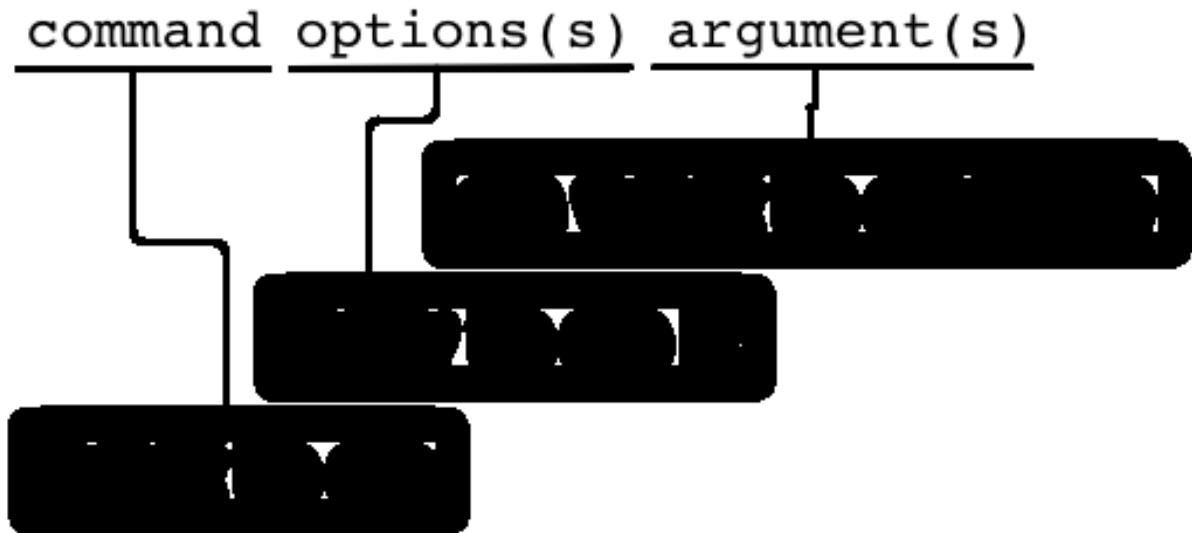
Figure 1.2: General structure of Linux commands.

Commandline options (sometimes called comandline switches) commonly have one of the two following forms: The short form `-s` (just a single character) or the long form `--string`. E.g.

```
$ man -h
$ man --help
```

Short options are usually - though not always - concatenable:

```
$ ls -l -A -h
$ ls -lAh
```

Some options require an additional argument, which is added with a blank to the short form and with an equal sign to the long form:

```
$ ls -I "*.pdf"
$ ls --ignore="*.pdf"
```

Since Linux incorporates commands from different sources, options can be available in one or both forms and you'll also encounter options with no dash at all and all kinds of mixtures:

```
$ tar cf file.tar -C .. file/
$ ps auxgww
```

1.4   A Journey Through the Commands

Please note that all examples and usage instructions below are just a glimpse of what you can do and reflect our opinion on what's important and what's not. Most of these commands support many more options and different usages. Consult the manpages to find them.

Typographical conventions: Commands and examples are written in Courier. User Input is written in Courier bold and placeholders are generally written in italic.

1.4.1 Useful Terminal Tools & Keyboard Shortcuts

Navigating previous commands

You can use the ↑/↓ (up/down) arrow keys to navigate previously entered command and the ←/→ (left/right) keys to modify it before re-executing it.

Copying / Pasting using the mouse

On most Linux systems you can use the mouse to select text and then press the middle mouse button to paste that text at the position where your cursor is. This is especially useful for long directory or filenames.

Printing some text

To simply print some text in the console, use `echo`:

Usage: `echo`

```
$ echo "this is some text"
this is some text
$
```

It can also be used to print the content of a variable, see section *Environment Variables* (*introduction:environment-variables)...

Interrupting commands

Whenever a program gets stuck or takes too long to finish, you can *interrupt* it with the shortcut `CONTROL-C`.

clear – Clear the "screen"

Usage: `clear`

```
$ clear
$
```

In case the output of the terminal/screen gets cluttered, you can use `clear` to redraw the screen...

If this doesn't work, you can use `reset` to perform a re-initialization of the terminal:

reset – Reset your terminal

Usage: `reset [options]`

```
$ reset
$
```

Leave the shell

To exit the shell/terminal, just type `exit` or press `CONTROL-D`.

1.4.2   Getting Help

`-h`/`--help` option, no parameters

Many commands support a "help" option, either through `-h` or through `--help`. Other commands will show a help page or at least a short usage overview if you provide incorrect commandline options

man - show the manual page of a command

Usage: `man command or file`

```
$ man man
man(1)

NAME
 man - format and display the on-line manual pages

SYNOPSIS
 man [-acdfFhkKtwW] [--path] [-m system] [-p string] [-C config_file]
...
$
```

For the navigation within a manpage see the *paragraph regarding less* (*introduction:less).

Note: The behaviour of `man` is dependent of the `$PAGER` environment variable

---

apropos - list manpages containing a keyword in their description

Usage: `apropos keyword`

```
$ apropos who
...
who                  (1)  - show who is logged on
who                  (1)  - display who is on the system
whoami               (1)  - print effective userid
$
```

:index *Use <apropos>* `apropos` to find candidates for specific tasks

/usr/share/doc/

The `/usr/share/doc/` directory in some Linux distributions contains additional documentation of installed software packages

---

1.4.3 Who am I, where am I

whoami - Print your username

Usage: `whoami`

```
$ whoami
fthommen
$
```

hostname - Print the name of the computer

Usage: `hostname`

```
$ hostname
pc-teach01
$
```

pwd - Print the current working directory

Usage: `pwd`

```
$ pwd
/home/fthommen
$
```

date - Print current date and time

Usage: `date`

```
$ date
Tue Sep 25 19:57:50 CEST 2012
$
```

Note: The command `time` does something completely different from `date` and is *not* used to show the current time.

1.4.4 Moving Around

cd - Change the working directory

Usage: `cd [new_directory]`

```
$ pwd
/home/fthommen
$ cd /usr/bin
$ pwd
/usr/bin
$
```

Special directories:

- ".": The current working directory

- "..": The parent directory of the current working directory

- "~": Your homedirectory

Note: Using `cd` without a directory is equivalent to "`cd ~`" and changes into the users's homedirectory

---

Note: Please note the difference between absolute pathes (starting with "/") and relative pathes (starting with a directory name)

```
$ pwd
/usr
$ cd /bin
$ pwd
/bin

$ pwd
/usr
$ cd bin
$ pwd
/usr/bin
$
```

1.4.5   See What's Around

ls - List directory contents

Usage: `ls [options] [file(s) or directory/ies]`

```
$ ls
/home/fthommen
$ ls -l aa.pdf
-rw-r--r-- 1 fthommen cmueller 0 Sep 24 10:59 aa.pdf
$
```

Useful options:

-l	Long listing with permissions, user, group and last modification date
-1	Print listing in one column only
-a	Show all files (hidden, "." and "..")
-A	Show almost all files (hidden, but not "." and "..")
-F	Show filetypes (nothing = regular file, "/" = directory, "*" = executable file, "@" = symbolic link)
-d	Show directory information instead of directory content
-t	Sort listing by modification time (most recent on top)
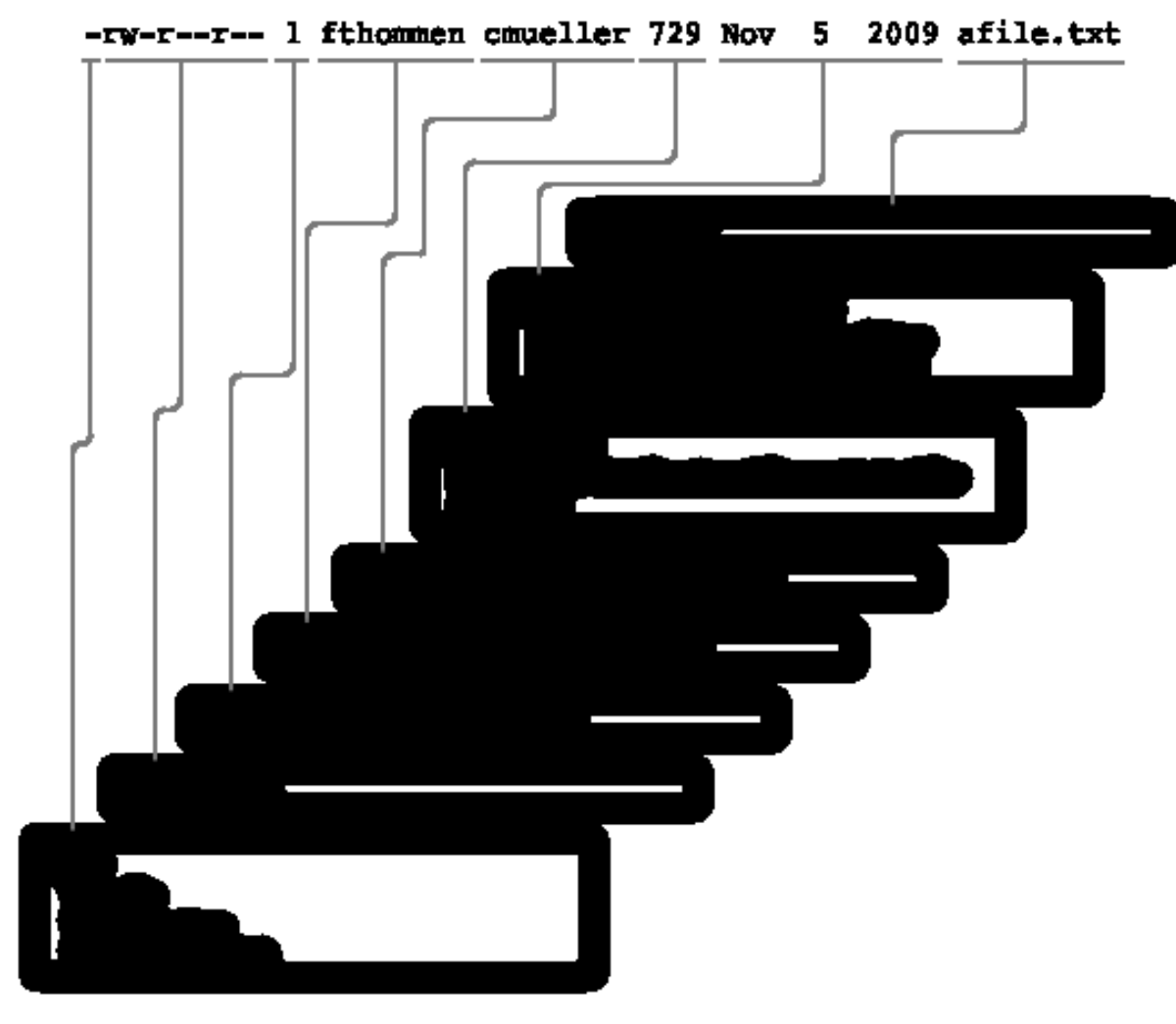
Digression: Shell globs

Files and folders can't only be referred to with their full name, but also with so-called "Shell Globs", which are a kind of simple pattern to address groups of files and folders. Instead of explicit names you can use the following placeholders:

- `?`: Any single character
- `*`: Any number of any character (including no character at all, but **not** matching a starting ".")
- `[...]`: One of the characters included in the brackets. Use "-" to define ranges of characters
- `{word1,word2}`: Each individual word is expanded

Examples:

- `*.pdf`: All files having the extension ".pdf"
- `?.jpg`: Jpeg file consisting of only one character
- `[0-9]*.txt`: All files starting with a number and having the extension ".txt"

- `*.???`: All files having a three-character extension

- `photo.{jpg,png}`: "photo.jpg" and "photo.png"

Note: The special directory "`~`" mentioned above is a shell glob, too.

1.4.6 Organize Files and Folders

cp – Copy files and folders

Usage: `cp [options] sourcefile destinationfile`

```
$ cp /usr/bin/less /tmp/backup_of_less
$
```

Useful options:

`-r`	Copy recursively
`-i`	Interactive operation, ask before overwriting an existing file
`-p`	Preserve owner, permissions and timestamp

We copy a set of exercise files from the network share into our home directory:

```
$ cp -r /g/bio-it/courses/LSB/exercises  ~/exercises
$
```

rsync - intelligently copying files and folders

Usage: `rsync [options] source target`

```
$ rsync -av /etc/ root@taperobot:/etc-backup
...
$
```

`rsync` allows to copy files or folders locally or to wherever you have `ssh` access. You can have `rsync` have copying only newer files or only older files. If copy operation is interrupted, you can rerun `rsync` and it will only copy the missing files (in contrast to `cp` which will just copy everything again).

`source` and `target` can be local directories or have the form user@remotehost:directory, in which case you'll have to give your password for the remote host. This latter version will copy over the network.

Note: `rsync` is one of the few cases, where it matters if a directory is written with an ending slash ("/") or nor: If the source is a directory and ends with a slash, then the **content** of this directory will be copied. If the source doesn't have an ending slash, then a directory with the same name will be created within the target directory

---

Useful option combinations:

-av		Verbosely copies all source files wich are different (different size, different age) or missing from the source. **Beware**: This will also copy files which are older on the source side
-au		Silently copies all source files wich are different (different size, different age) or missing from the source. This combination will not overwrite newer files by older ones

touch – Create a file or change last modification date of an existing file

Usage: `touch file(s) or directory/ies`

```
$ ls afile
ls: afile: No such file or directory
$ touch afile
$ ls afile
afile
$

$ ls -l aa.pdf
-rw-r--r-- 1 fthommen cmueller 0 Sep 24 10:59 aa.pdf
$ touch aa.pdf
$ ls -l aa.pdf
-rw-r--r-- 1 fthommen cmueller 0 Sep 25 22:01 aa.pdf
$
```

rm - Remove files and directories

Usage: rm [options] file(s)

```
         rm -r [options] directory/ies
```

```
$ ls afile
afile
$ rm afile
$ ls afile
ls: afile: No such file or directory
$
```

Useful options:

-i		Ask for confirmation of each removal
-r		Remove recursively
-f		Force the removal (no questions, no errors if a file doesn't exist)

Note: rm without the -i option will usually not ask you if you really want to remove the file or directory

mv - Move and rename files and folders

Usage: mv [options] sourcefile destinationfile

```
      mv [options] sourcefile(s) destinationdirectory
```

```
$ ls *.txt
a.txt
$ mv a.txt b.txt
$ ls *.txt
b.txt
$
```

Useful options:

> **-i**   Ask for confirmation of each removal

Note: You cannot overwrite an existing directory by another one with mv

---

mkdir - Create a new directory

Usage: `mkdir [options] directory`

```
$ ls adir/
ls: adir/: No such file or directory
$ mkdir adir
$ ls adir
$
```

Useful options:

> **-p**   Create parent directories (when creating nested directories)

```
$ mkdir adir/bdir
mkdir: cannot create directory 'adir/bdir': No such file or directory
$ mkdir -p adir/bdir
$
```

rmdir - Remove an empty directory

Usage: `rmdir directory`

```
$ rmdir adir/
$
```

Note: If the directory is not empty, `rmdir` will complain and not remove it.

1.4.7   View Files

cat - Print files on terminal (concatenate)

Usage: `cat [options] file(s)`

```
$ cat  P12931.fasta backup_of_P12931.fasta
...
$
```

Note: The command `cat` only makes sense for short files or for e.g. combining several files into one. See the redirection examples later.

---

head - Print first lines of a textfile

`head` is a program on Unix and Unix-like systems used to display the beginning of a text file or piped data.

Usage: `head [options] file(s)`

```
$ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
news:x:9:13:news:/etc/news:
$
```

Useful options:

-n num	Print num lines (default is 10)

tail - Print last lines of a textfile

The `tail` utility displays the contents of file or, by default, its standard input, to the standard output.

Usage: `tail [options] file(s)`

```
$ tail -n 3 /etc/passwd
xfs:x:43:43:X Font Server:/etc/X11/fs:/sbin/nologin
gdm:x:42:42::/var/gdm:/sbin/nologin
sabayon:x:86:86:Sabayon user:/home/sabayon:/sbin/nologin
$
```

Useful options:

-n num	Print num lines (default is 10)
-f	"Follow" a file (print new lines as they are written to the file)

less - View and navigate files

Usage: `less [options] file(s)`

```
$ less  P12931.fasta backup_of_P12931.fasta
...
$
```

Note: This is the default "pager" for manpages under Linux unless you redefine your `$PAGER`
environment variable (*introduction:environment-variables)

Navigation within less:

Key(s):	Effect:
up, down, right, left:	use cursor keys
top of document:	g
bottom of document:	G
search:	"/" + search-term
find next match:	n
find previous match:	N
quit:	q

1.4.8 Extracting Informations from Files

grep - Find lines matching a pattern in textfiles

`grep` is a command-line utility for searching plain-text data sets for lines matching a regular
expression.

Usage: `grep [options] pattern file(s)`

```
$ grep -i ensembl P04637.txt
DR   Ensembl; ENST00000269305; ENSP00000269305; ENSG00000141510.
DR   Ensembl; ENST00000359597; ENSP00000352610; ENSG00000141510.
DR   Ensembl; ENST00000419024; ENSP00000402130; ENSG00000141510.
DR   Ensembl; ENST00000420246; ENSP00000391127; ENSG00000141510.
DR   Ensembl; ENST00000445888; ENSP00000391478; ENSG00000141510.
DR   Ensembl; ENST00000455263; ENSP00000398846; ENSG00000141510.
$
```

Useful options:

-v Print lines that do not match

-i Search case-insensitive

-l List files with matching lines, not the lines itself

-L List files without matches

-c Print count of matching lines for each file

cut - extracting columns from textfiles

`cut` allows to get at individual columns in structured textfiles (for instance CSV files). By
default, `cut` assumes the columns are TAB-separated.

Usage: `cut [options] file(s)`

Useful options:

-d DELIM use DELIM instead of TAB for field delimiter. Make sure to use
quotes here!

-f	select only these fields; this can either be a single field, multiple individual fields separated by comma or a range of startfield and endfield separated by dash '-'

Examples:

extract column six from the file */exercises/P12931.csv* (which is separated by semicolon ';'):

```
$ cut -d';' -f6 /exercises/P12931.csv
PMID
2136766
11804588
...
$
```

extract columns two, three, eight, nine and ten from the same file:

```
$ cut -d';' -f2,3,8-10 /exercises/P12931.csv
S; 12; 0.21; ; -
S; 17; 0.24; MOD_PKA_1; -
S; 17; 0.24; MOD_PKA_1; -
S; 17; 0.24; MOD_PKA_1; -
...
$
```

sort - sort a textfile

The `sort` utility is used to sort a textfile (alphabetically or numerically).

Usage: `sort [options] file(s)`

```
$ sort /etc/passwd
...
$
```

Useful options:

-f	fold lower case to upper case characters
-n	compare according to string numerical value
-b	ignore leading blanks
-r	reverse the result of comparisons

1.4.9   Useful Filetools

file - determine the filetype

Usage: `file [options] file(s)`

```
$ file /bin/date
/bin/date: ELF 32-bit LSB executable
$ file /bin
/bin: directory
```

```
$ file SRC_HUMAN.fasta
SRC_HUMAN.fasta: ASCII text
$
```

Note: The command `file` uses certain tests and some magic to determine the type of a file

---

which - find a (executable) command

Usage: `which [options] command(s)`

```
$ which date
/bin/date
$ which eclipse
/usr/bin/eclipse
$
```

find - search/find files in any given directory

Usage: `find [starting path(s)] [search filter]`

```
$ find /etc
/etc
/etc/printcap
/etc/protocols
/etc/xinetd.d
/etc/xinetd.d/ktalk
...
$
```

`find` is a powerful command with lots of possible search filters. Refer to the manpage for a complete list.

Examples:

- Find by name:

```
$ find . -name SRC_HUMAN.fasta
./SRC_HUMAN.fasta
$
```

- Find by size: (List those entries in the directory **/usr/bin** that are bigger than 500kBytes)

```
$ find /usr/bin -size +500k
/usr/bin/oparchive
/usr/bin/kiconedit
/usr/bin/opjitconv
...
$
```
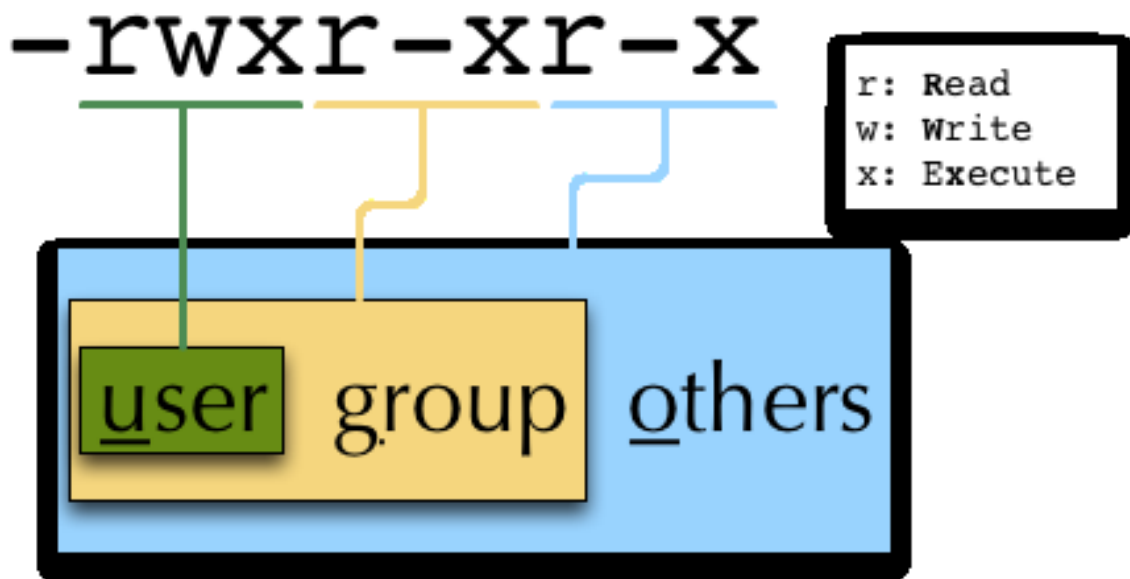
- Find by type (d=directory, f=file, l=link)

```
$ find . -type d
.
./adir
$
```

1.4.10 Permissions

using `ls -l` to view entries of current directory:

```
$ ls -l
drwxr-xr-x 2 dinkel gibson 4096 Sep 17 10:46 adir
lrwxrwxrwx 1 dinkel gibson   15 Sep 17 10:45 H1.fasta -> H2.fasta
-rw-r--r-- 1 dinkel gibson  643 Sep 17 10:45 H2.fasta
$
```


Changing Permissions

Permissions are set using the `chmod` (change mode) command.

Usage: `chmod [options] mode(s) files(s)`

```
$ ls -l adir
drwxr-xr-x 2 dinkel gibson 4096 Sep 17 10:46 adir
$ chmod u-w,o=w adir
$ ls -l adir
dr-xr-x-w- 2 dinkel gibson 4096 Sep 17 10:46 adir
$
```

The mode is composed of

Who		What		Which permission	
u:	user/owner	+:	add this permission	r:	read
g:	group	-:	remove this permission	w:	write
o:	other	=:	set exactly this permission	x:	execute
a:	all				

Add executable permission to the group:

```
$ chmod g+x file
$
```

Revoke this permission:

```
$ chmod g-x file
$
```

Allow all to read a directory:

```
$ chmod a+rx adir/
$
```

1.4.11   Remote access

To execute commands at a remote machine/server, you need to log in to this machine. This is done using the `ssh` command (secure shell). In its simplest form, it takes just the machinename as parameter (assuming the username on the local machine and remote machine are identical):

```
$ ssh remote_server
...
$
```

Note: Once logged in, use `hostname, whoami,` etc. to determine on which machine you are currently working and to get a feeling for your environment!

To use a different username, you can use either:

```
$ ssh -l username remote_server
...
$
```

or

```
$ ssh username@remote_server
...
$
```

When connecting to a machine for the first time, it might display a warning:

```
$ ssh sub-master
The authenticity of host 'sub-master (10.11.4.84)' can't be established.
RSA key fingerprint is 47:a4:0f:7b:c2:0f:ef:91:8e:65:fc:3c:f7:0c:53:8d.
Are you sure you want to continue connecting (yes/no)?
...
$
```

Type *yes* here. If this message appears a second time, you should contact your IT specialist...

To disconnect from the remote machine, type:

```
$ exit
```

If setup correctly, you can even use **graphical tools** from the remote server on the local machine. For this to work, you need to start the ssh session with the `-X` parameter:

```
$ ssh -X remote_server
...
$
```

Copying files to and from remote computers can be done using `scp` (secure copy). The order of parameters is the same as in `cp`: first the name of the source, then the name of the destination. Either one can be the remote part.

```
$ scp localfile server:/remotefile
```

```
$ scp server:/remotefile localfile
```

An alternative username can be provided just as in ssh:

```
$ scp username@server:/remotefile localfile
```

1.4.12   IO and Redirections

Redirect

Redirect the output of one program into e.g. a file:

Inserting the current date into a new file:

```
$ date > file_containing_date
$
```

Warning: You can easily overwrite files by this!

Filtering lines containing the term "src" from FASTA files and inserting them into the file lines_with_src.txt:

```
$ cd /exercises/
$ grep -i "src" *.fasta > lines_with_src.txt
$
```

Append

Append something to a file (rather than overwriting it):

```
$ date >> file_containing_date
$
```

Pipe

Use the pipe symbol (|) to feed the output of one program into the next program. Here: use `ls` to show the directory contents and then use `grep` to only show those that contain fasta in their name:

```
$ cd /exercises
$ ls | grep fasta
EPSINS.fasta
FYN_HUMAN.fasta
P12931.fasta
SRC_HUMAN.fasta
$
```

1.4.13 Environment Variables

Environment variables are a set of dynamic named values that can affect the way running processes will behave on a computer.

$HOME

Contains the location of the user's home directory. Although the current user's home directory can also be found out through the C functions `getpwuid` and `getuid,` $HOME is often used for convenience in various shell scripts (and other contexts).

Note: Do not change this variable unless you have a good reason and you know what you are doing!

$PATH

$PATH contains a colon-separated (':') list of directories that the shell searches for commands that do not contain a slash in their name (commands with slashes are interpreted as file names to execute, and the shell attempts to execute the files directly). So if the directory */usr/bin/* is in *$PATH* (which it should), then the command **/usr/bin/less** can be accessed by simply typing `less` instead of **/usr/bin/less**. How convenient!

> **Warning:** If you ever need to change this variable, you should always **append** to it, rather than overwriting it:
> Overwriting (bad): `export PATH=/my/new/path`;
> Appending (good): `export PATH=$PATH:/my/new/path`

$PAGER

The $PAGER variable contains the path to the program used to list the contents of files through (such as `less` or `more`).

$PWD

The $PWD variable points to the current directory. Equivalent to the output of the command `pwd` when called without arguments.

Displaying environment variables

Use `echo` to display individual variables *set'* or `env` to view all at once:

```
$ echo $HOME
/localhome/teach01
$ set
...
$ env
```

```
...
$
```

Setting an environment variable

Use `export` followed by the variable name and the value of the variable (separated by the equal sign) to set an environment variable:

```
$ export PAGER=/usr/bin/less
$
```

Note: An environment variable is only valid for your current session. Once you logout of your current session, it is lost or reset.

Chapter 2

Exercises

2.1 Misc. file tools

1. Which tool can be used to determine the type of a file?
2. Use it on the following files/directories and compare the results:

    (a) `/usr/bin/tail`

    (b) `~`

    (c) `/exercises/SRC_HUMAN.fasta`

2.2 Searching

1. Which tool can be used to search for files or directories?
2. Use it to find all directories in the `/exercises` directory
3. Search for the file date in the `/bin` directory
4. List those entries in the directory `/bin` that are bigger than 400kBytes

2.3 Misc. terminal

1. Which two tools can be used to redraw/empty the screen?

2.4 Permissions

1. Create a directory called testpermissions
2. Change your working directory to testpermissions
3. Create a directory called adir.
4. Use the command `which date` to find out where the date program is located.

5. Copy this date program into the directory adir.

6. Check the permissions of the copied program date

7. Change the permissions on date to remove the executable permissions.

8. Check the permissions of the program date

9. Try running it as `./date` or `adir/date` (depending on your current working directory)

10. Change the permissions back so that the file is executable.

11. Try running it as `./date` or `adir/date` (depending on your current working directory)

12. Copy a textfile from a previos exercise into adir, then change the permissions, so you are not allowed to write to it.

13. Then change the permissions so you can't read/cat it either.

14. Change your working directory to testpermissions, and then try changing the permissions on the directory adir to non-executable.

15. What are the minimum permissions (on the directory) necessary for you to be able to execute `adir/date`?

2.5   Remote access

1. Login to machine "sub-master.embl.de" (using your own username)

2. Use exit to quit the remote shell (Beware to not exit your local shell)

3. Use clear to empty the screen after logout from the remote server

4. Use the following commands locally as well as on the remote machine to get a feeling for the different machines:

    (a) `hostname`

    (b) `whoami`

    (c) `ls -la ~/`

5. Copy the file `/etc/motd` from machine sub-master.embl.de into your local home directory (using scp)

6. Determine the filetype and the permissions of the file that you just copied

7. Login to your neighbor's machine (ask him for the hostname) using your own username

2.6   IO and Redirections

1. Use date in conjuction with the redirection to insert the current date into the (new) file current_date (in your homedirectory).

2. Inspect the file to make sure it contains (only a single line with) the date.

3. Use `date` again to append the current date into the same file.

4. Again, check that this file now contains two lines with dates.

5. Use `grep` to filter out lines containing the term "TITLE" from all PDB files in the exercises directory and use redirection to insert them into a new file pdb_titles.txt.

6. (OPTIONAL) Upon inspection of the file pdb_titles.txt, you see that it also contains the names of the files in which the term was found.

   (a) Use either the grep manpage or `grep --help` to find out how you can suppress this behaviour.

   (b) Redo the previous exercise such that the output file pdb_titles.txt only contains lines starting with TITLE.

7. The *third* column of the file /etc/passwd/ contains user IDs (numbers)

   (a) Use *cut* to extract just the third column of this file (remember to specify the delimiter ':'):

   (b) Next, use the *pipe* (*introduction:pipe) symbol (/) and *sort* to sort this output *numerically*:

Chapter 3

Links and Further Information

3.1 Links

- A full 500 page book about the Linux commandline for free(!): LinuxCommand.org [1]

- Another nice introduction: "A beginner's guide to UNIX/Linux" [2]

- The "*commandline starter*" chapter of an O'Reilly book: Learning Debian GNU/Linux - Issuing Linux Commands [3]

- A nice introduction to Linux/UNIX file permissions: "chmod Tutorial" [4]

- Linux Cheatsheets [5]

- For the technically interested: Linux Filesystem Hierarchy Standard [6] and Linux Standard Base [7]

- Unix commands applied to bioinformatics [8]

- BioPieces [9] are a collection of bioinformatics tools that can be pieced together in a very easy and flexible manner to perform both simple and complex tasks.

- Google shell style guide [10]

- Useful bash one-liners for bioinformatics [11]

- Interactive explanation of your commandline: Explain Shell [12]

---

[1] http://linuxcommand.org/
[2] http://www.mn.uio.no/astro/english/services/it/help/basic-services/linux/guide.html
[3] http://oreilly.com/openbook/debian/book/ch04_01.html
[4] http://www.catcode.com/teachmod/
[5] http://www.cheat-sheets.org/#linux
[6] http://www.pathname.com/fhs/
[7] http://www.linuxfoundation.org/collaborate/workgroups/lsb
[8] http://rous.mit.edu/index.php/Unix_commands_applied_to_bioinformatics
[9] http://code.google.com/p/biopieces
[10] https://code.google.com/p/google-styleguide
[11] https://github.com/stephenturner/oneliners
[12] http://www.explainshell.com

3.2 Command Line Mystery Game

CLMystery [13] is a game that you play on the commandline: There's been a murder in Terminal City, and TCPD needs your help to solve this crime *by using commandline tools only*!

To play the game, get the files from github and read the instructions:

```
wget https://github.com/veltman/clmystery/archive/master.zip
unzip master.zip
cd clmystery-master/
cat instructions
```

3.3 Real printed paper books

- Dietz, M., "*Praxiskurs Unix-Shell*", O'Reilly (highly recommended!, german language only)

- Herold, H., "*awk & sed*", Addison-Wesley

- Robbins, A., "*sed & awk Pocket Reference*", O'Reilly

- Robbins, A. and Beebe, N., "*Classic Shell Scripting*", O'Reilly

- Siever, E. et al., "*Linux in a Nutshell*", O'Reilly

3.4 Live - CDs

A Live-CD is a complete bootable computer operating system which runs in the computer's memory, rather than loading from the hard disk drive. It allows users to experience and evaluate an operating system without installing it or making any changes to the existing operating system on the computer.

Just download an ISO-Image, burn it onto a CD/DVD and insert it into your DVD-Drive to boot your computer with Linux!

3.4.1 Fedora Live CD

This Live CD contains everything the Fedora [14] Linux operating system has to offer and it's everything you need to try out Fedora - you don't have to erase anything on your current system to try it out, and it won't put your files at risk. Take Fedora for a test drive, and if you like it, you can install Fedora directly to your hard drive straight from the Live Media desktop.

3.4.2 Knoppix

Knoppix [15] is an operating system based on Debian designed to be run directly from a CD / DVD or a USB flash drive, one of the first of its kind for any operating system. When starting a program, it is loaded from the removable medium and decompressed into a RAM drive. The

---

[13] https://github.com/veltman/clmystery
[14] http://fedoraproject.org/wiki/FedoraLiveCD
[15] http://knopper.net/knoppix

decompression is transparent and on-the-fly. More than 1000 software packages are included on the CD edition and more than 2600 are included on the DVD edition. Up to 9 gigabytes can be stored on the DVD in compressed form.

3.4.3 BioKnoppix

Bioknoppix [16] is a customized distribution of Knoppix Linux Live CD. With this distribution you just boot from the CD and you have a fully functional Linux OS with open source applications targeted for the molecular biologist. Beside using RAM, Bioknoppix doesn't touch the host computer, being ideal for demonstrations, molecular biology students, workshops, etc.

3.4.4 Vigyaan

Vigyaan [17] is an electronic workbench for bioinformatics, computational biology and computational chemistry. It has been designed to meet the needs of both beginners and experts.

3.4.5 BioSlax

BioSLAX [18] is a live CD/DVD suite of bioinformatics tools that has been released by the resource team of the BioInformatics Center (BIC), National University of Singapore (NUS).

---

[16] http://bioknoppix.hpcf.upr.edu
[17] http://www.vigyaancd.org
[18] http://www.bioslax.com

Chapter 4

About Bio-IT

Bio-IT is a community project aiming to develop and strengthen the bioinformatics user community at EMBL Heidelberg. It is made up of members across the different EMBL Heidelberg units and core facilities. The project works to achieve these aims, firstly, by providing a forum for discussing and sharing information and ideas on computational biology and bioinformatics, focused on the Bio-IT portal. Secondly, we organise and participate in a range of different networking and social activites aiming to strengthen ties across the community.

4.1   Resources

A list of biocomputing-related resources associated with the project, in the top-left "Resources" menu, including, for example there is help provided for installing software on Linux computers at EMBL, instructions on using the Git versions control system server provided by EMBL, and various other kinds of information.

4.2   Training and Outreach

The "Training and Outreach" menu, bottom left, provides information on events (courses and conferences), both internal to EMBL and organised elsewhere by other organisations, that are related to biocomputing and bioinformatics

4.3   Networking

Several different kinds of networking events for the Bio-IT community are being organised, including beer sessions for the EMBL community, and within-Heidelberg events for the larger Heidelberg biocomputing community.

4.4   Biocomputing expertise at EMBL

You can use the Bio-IT portal to search for people working at EMBL who have experience working with data or tools you might be interested in.

If you've not yet got a page up on the portal describing your own expertise, please do so. If you need any help doing this, you can read about this in the portal's FAQ section, or get in touch with one of the site administrators.

Chapter 5

Acknowledgements

EMBL Logo © EMBL Heidelberg

Graphic of the *Linux Filesystem* (*introduction:figure-filesystem) taken from the SuSE 9.2 manual © Novell Inc.

All other graphics © Frank Thommen, EMBL Heidelberg, 2012 **License**: CC BY-SA 3.0

Chapter 6

Solutions to the Exercises

6.1 Misc. file tools

1. Which tool can be used to determine the type of a file?

   ```
   $ file
   ```

2. Use it on the following files/directories and compare the results:

   (a) `/usr/bin/tail`

   ```
   $ file /usr/bin/tail
   /usr/bin/tail: binary executable
   ```

   (b) `~`

   ```
   $ file ~
   /home/dinkel: directory
   ```

   (c) `/exercises/SRC_HUMAN.fasta`

   ```
   $ file /exercises/SRC_HUMAN.fasta
   /exercises/SRC_HUMAN.fasta: ASCII text
   ```

6.2 Searching

1. Which tool can be used to search for files or directories?

   ```
   $ find
   ```

2. Use it to find all directories in the `/exercises` directory

   ```
   $ find /exercises -type d
   ```

3. Search for the file date in the `/bin` directory

   ```
   $ find /bin -name date
   ```

4. List those entries in the directory `/bin` that are bigger than 400kBytes

```
$ find /bin -size +400k
```

6.3   Misc. terminal

1. Which two tools can be used to redraw/empty the screen?

   ```
   $  clear
   ```

6.4   Permissions

1. Create a directory called testpermissions

   ```
   $ mkdir testpermissions
   ```

2. Change your working directory to testpermissions

   ```
   $ cd testpermissions
   ```

3. Create a directory called adir.

   ```
   $ mkdir adir
   ```

4. Use the command `which date` to find out where the date program is located.

   ```
   $ which date
   /bin/date
   ```

5. Copy this date program into the directory adir.

   ```
   $ cp /bin/date adir/
   ```

6. Check the permissions of the copied program date

   ```
   $ ls -lh adir/date
   -r-xr-xr-x  1 dinkel  staff    79K  9 Dec 13:47 date*
   ```

7. Change the permissions on date to remove the executable permissions.

   ```
   $ chmod a-x adir/date
   ```

8. Check the permissions of the program date

   ```
   $ ls -lh adir/date
   -r--r--r--  1 dinkel  staff    79K  9 Dec 13:47 date*
   ```

9. Try running it as `./date` or `adir/date` (depending on your current working directory)

   ```
   $ adir/date
   permission denied
   ```

10. Change the permissions back so that the file is executable.

    ```
    $ chmod a+x adir/date
    ```

11. Try running it as `./date` or `adir/date` (depending on your current working directory)

    ```
    $ adir/date
    Mon Dec  9 13:50:12 CET 2013
    ```

12. Copy a textfile from a previos exercise into adir, then change the permissions, so you are not allowed to write to it.

    ```
    $ cp /exercises/SRC_HUMAN.fasta adir
    $ chmod u-w adir/SRC_HUMAN.fasta
    ```

13. Then change the permissions so you can't read/cat it either.

    ```
    $ chmod u-r adir/SRC_HUMAN.fasta
    ```

14. Change your working directory to testpermissions, and then try changing the permissions on the directory adir to non-executable.

    ```
    $ # we don't need to change directory, because we still are in the directory testpermissions
    $ chmod a-x adir
    ```

15. What are the minimum permissions (on the directory) necessary for you to be able to execute `adir/date`?

    ```
    $ chmod u+rx adir
    ```

6.5 Remote access

1. Login to machine "sub-master.embl.de" (using your own username)

   ```
   $ ssh sub-master.embl.de -l username
   ```

2. Use exit to quit the remote shell (Beware to not exit your local shell)

   ```
   $ exit
   ```

3. Use clear to empty the screen after logout from the remote server

   ```
   $ clear
   ```

4. Use the following commands locally as well as on the remote machine to get a feeling for the different machines:

   A) ``hostname``

   B) ``whoami``

   C) ``ls -la ~/``

5. Copy the file `/etc/motd` from machine sub-master.embl.de into your local home directory (using scp)

   ```
   $ scp sub-master.embl.de:/etc/motd ~/
   ```

6. Determine the filetype and the permissions of the file that you just copied

```
$ file ~/motd
~/motd: ASCII text
```

```
$ ls -l ~/motd
```

7. Login to your neighbor's machine (ask him for the hostname) using your own username

   ```
   $ ssh hostname
   ```

6.6   IO and Redirections

1. Use date in conjuction with the redirection to insert the current date into the (new) file current_date (in your homedirectory).

   ```
   $ date > ~/current_date
   ```

2. Inspect the file to make sure it contains (only a single line with) the date.

   ```
   $ cat ~/current_date
   ```

3. Use `date` again to append the current date into the same file.

   ```
   $ date >> ~/current_date
   ```

4. Again, check that this file now contains two lines with dates.

   ```
   $ cat ~/current_date
   ```

5. Use `grep` to filter out lines containing the term "TITLE" from all PDB files in the exercises directory and use redirection to insert them into a new file pdb_titles.txt.

   ```
   $ grep TITLE /exercises/*.pdb > pdb_titles.txt
   ```

6. (OPTIONAL) Upon inspection of the file pdb_titles.txt, you see that it also contains the names of the files in which the term was found.

   (a) Use either the grep manpage or `grep --help` to find out how you can suppress this behaviour.

   ```
   $ grep -h TITLE /exercises/*.pdb > pdb_titles.txt
   ```

   (b) Redo the previous exercise such that the output file pdb_titles.txt only contains lines starting with TITLE.

   ```
   $ grep -h "^TITLE" /exercises/*.pdb > pdb_titles.txt
   ```

7. The *third* column of the file /etc/passwd/ contains user IDs (numbers)

   (a) Use *cut* to extract just the third column of this file (remember to specify the delimiter ':'):

   ```
   $ cut -f3 -d':' /etc/passwd
   ```

   (b) Next, use the *pipe* (*introduction:pipe) symbol (/) and *sort* to sort this output *numerically*:

```
$ cut -f3 -d':' /etc/passwd | sort -n
```