Introduction to the Linux Commandline

Holger Dinkel, Frank Thommen & Toby Hodges

October 16, 2015

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 correctly.
- 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 distributions 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 taught 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 ("/").

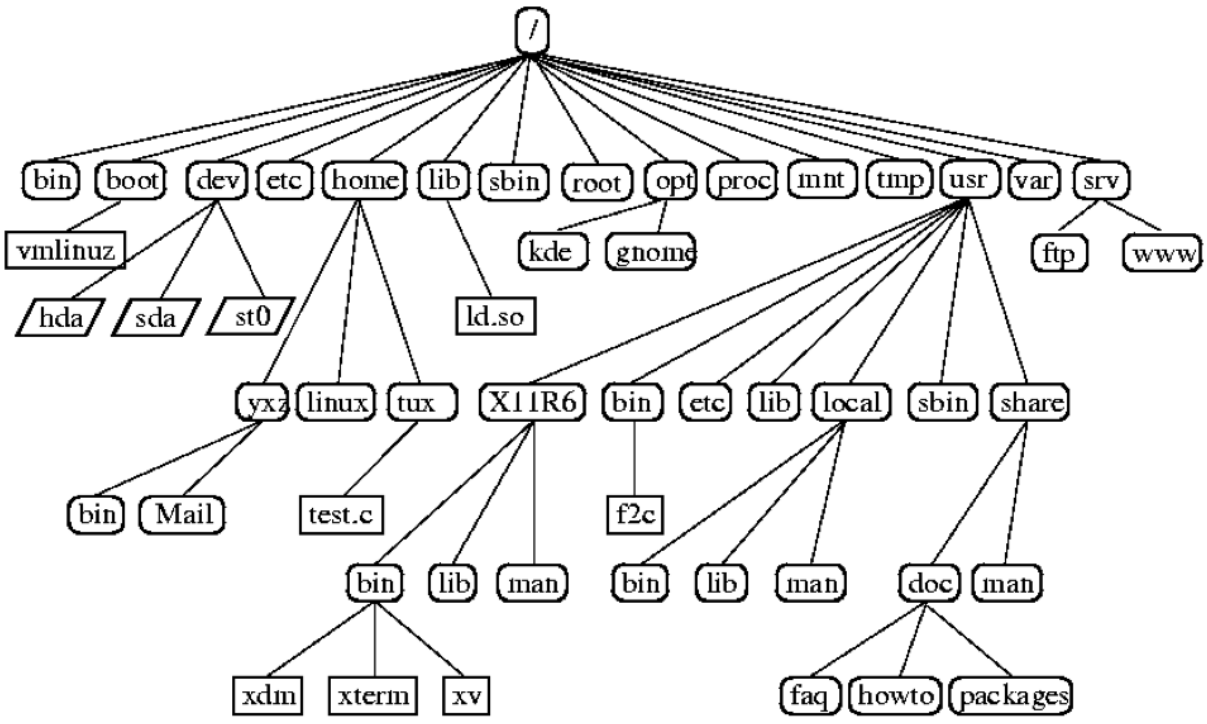

Figure 1.1: Depending on the Linux distribution you might or might not find all of the above directories. Most important directories for you are `/bin` and `/usr/bin` (sometimes also `/usr/local/bin`) which contain the user software, `/home` which usually contains the users' homedirectories and `/tmp` 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.

1.2.1 Absolute Paths / Relative Paths

A path describes the location of a file/folder in the filesystem: It is important to understand that there are basically two ways to describe such a path: Either by using an *absolute* pathname, or by using a *relative* pathname. The difference is that *absolute* paths always start with a "slash /". This "slash" denotes the so called "root" of the filesystem (see below). *Relative* paths in contrast always start with a directory name and denote the location of a file/folder *relative* to the current directory.

Note: When in doubt, it's best to use *absolute* filenames. Commands given with absolute pathname are more easily repeated later, as they can be run regardless of the current working directory (unlike relative paths).

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).

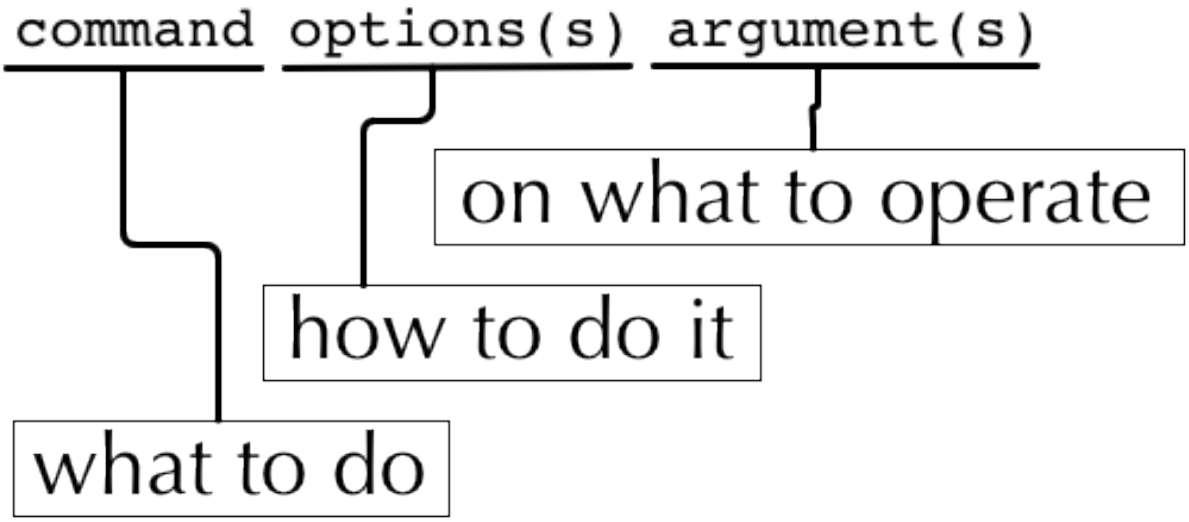

Figure 1.2: General structure of Linux commands.

For example, in the commandline `ls -l /usr/bin`, `ls` is the command, `-l` is an option and `/usr/bin` qualifies as an argument.

Commandline options (sometimes called commandline 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 - concatenatable:

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

Some options require an additional argument, which is added after a blank to the short form and 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.

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.

Saving time/avoiding typos with autocompletion

On most Linux systems you can autocomplete command names and filepaths by pressing `TAB`. This looks at the characters that you have entered so far and tries to predict what the rest of the command/path will be. This can save you from having to type out long command and file/directory names, and also reduces the likelihood of you accidentally spelling something incorrectly.

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* (page 23)...

Interrupting commands

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

Leave the shell

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

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...

```
$ cat /bin/echo
$ ...(garbled output here)
$ clear
$
```

Note: 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
$
```

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 echo
echo(1)

NAME
       echo - display a line of text

SYNOPSIS
       echo [SHORT-OPTION]... [STRING]...
       echo LONG-OPTION
  ...
$
```

For the navigation within a manpage and how to exit the manpage, see the *paragraph regarding less* (page 15).

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
$
```

Use `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

Linux is a multi-User Operating System supporting thousands of users on the same machine. As usernames can differ between machines, it's important to know your username on any particular machine.

Usage: `whoami`

```
$ whoami
fthommen
$
```

hostname - Print the name of the computer

Each machine on the network has a unique name which is used to distinguish one from another.

Usage: `hostname`

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

pwd - Print the current working directory

A Linux filesystem contains countless directories with many subdirectories which makes it easy to get lost. It is good practice to check your position within the filesystem regularly.

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
$
```

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

---

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

Special directories:

- ".": The current working directory
- "/": The root directory of this computer
- "..": The parent directory of the current working directory
- "~": Your homedirectory

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

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

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)

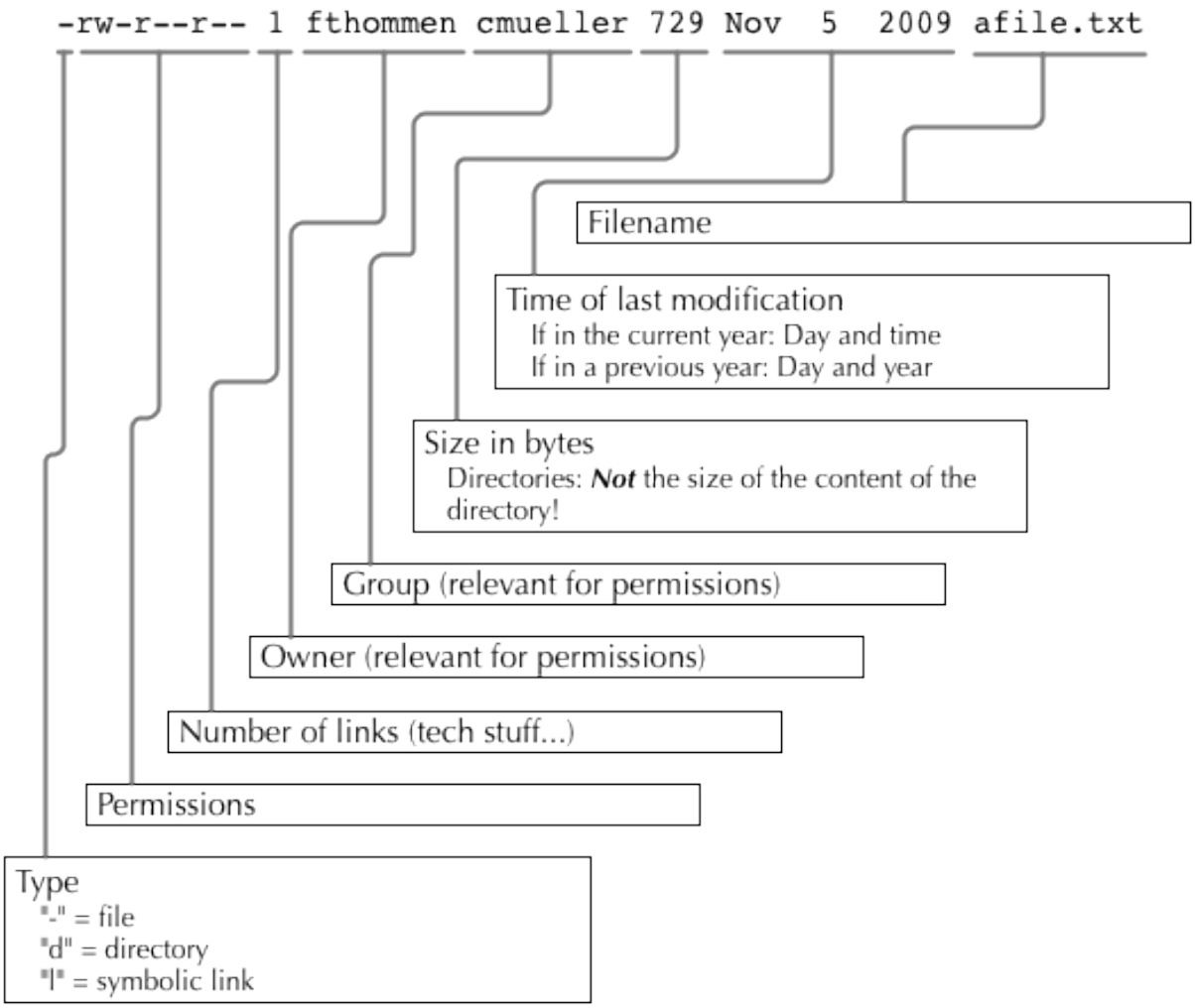

Figure 1.3: Elements of a long file listing (`ls -l`)

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

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
$
```

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

Examples:

If the last filename given is nonexisting then the first file is copied as this new filename:

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

Be careful! If the last filename given does exist, this file will be overwritten and replaced with a copy of the first file.

If the last filename given is an (existing!) directory, then the file is copied into this directory:

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

This allows us to copy multiple files into the same directory at the same time:

```
$ cp /usr/bin/less /usr/bin/grep /usr/bin/tail /tmp/
$
```

To recursively copy files, we need to specify the `-r` option. Here, we copy a set of exercise files from the network share into our home directory:

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

rsync - intelligently copying files and folders

Usage: `rsync [options] source target`

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

`rsync` allows you to copy files or folders locally or to wherever you have `ssh` access. You can have `rsync` copy 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 effectively 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 into the target directory. 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 which 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 which are different (different size, different age) or missing from the source. This combination will *not* overwrite newer files by older ones

This should not copy any new files, as we previously copied these already:

```
$ rsync -av /g/bio-it/courses/LSB/exercises/  ~/exercises/
$
```

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 last few lines of a 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" (a program for viewing files page by page, not an old-fashioned telecommunications device) for manpages under Linux unless you redefine your `$PAGER` *environment variable* (page 23)

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
-A NUM	print NUM lines of trailing context (After)
-B NUM	print NUM lines of leading context (Before)
-C NUM	print NUM lines of output context (Context)

Examples:

- List all files in the current directory which contain the searchterm `Ensembl`:

```
$ grep -l Ensembl ./*
P04637.txt
P12931.txt
```

Note: You cannot combine the option `-v` and `-l` to find files which do not contain a certain searchterm. The reason is that grep works line-based and not really file-based... Therefore you should rather use the uppercase `-L` option!

- List all files in the current directory which **do not** contain the searchterm `Ensembl`:

```
$ grep -L Ensembl ./*
1FMK.pdb
3A4O.pdb
...
```

- Count the number of occurrences (case insensitive!) of the term `atom` in all pdb files:

```
$ grep -ic atom ./*.pdb
```

- Find the term 'Homo sapiens' in the file P04637.txt, but also print two lines before the match:

```
$ grep -A2 'Homo sapiens' P04637.txt
```

- Find the term 'Homo sapiens' in the file P04637.txt, but also print the three lines following the match:

```
$ grep -B3 'Homo sapiens' P04637.txt
```

- Find the term 'Homo sapiens' in the file P04637.txt, but also print the surrounding five lines:

```
$ grep -C5 'Homo sapiens' P04637.txt
```

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 500 kBytes)

```
$ 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

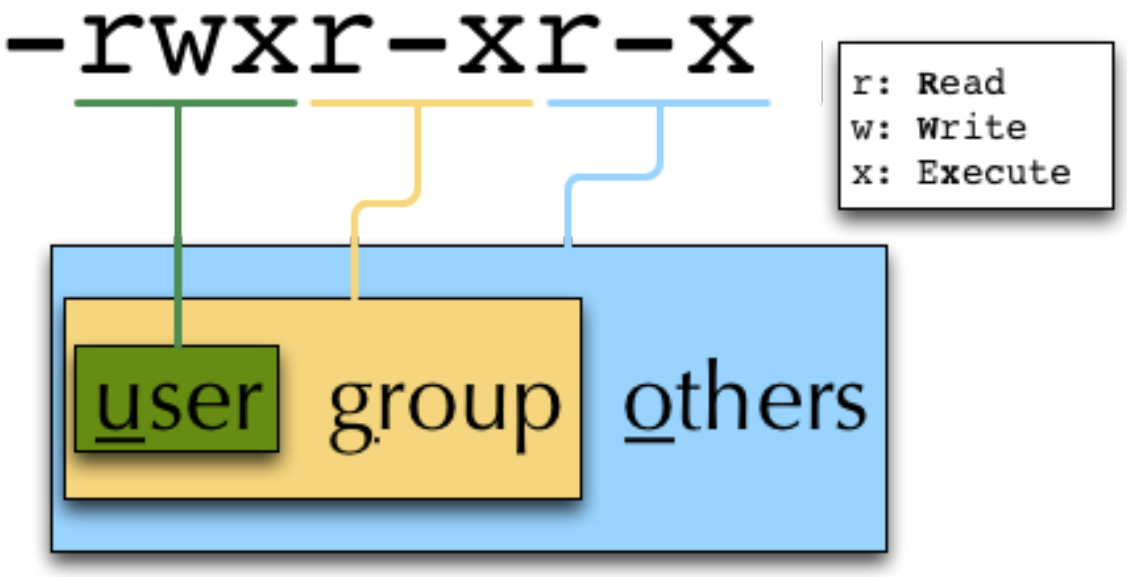

Figure 1.4: Linux file permissions

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 submaster
The authenticity of host 'submaster (10.11.4.219)' can't be established.
RSA key fingerprint is a4:2c:c1:a6:34:49:a3:a9:b2:c3:52:f5:37:94:69:f5.
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 Copying / Deleting Files & Folders

1. Navigate to your home directory
2. In your home directory, create a new directory named `new_dir`
3. Change into this directory, create a new empty file in there named `new_file`, and make sure that the file was created.
4. Duplicate this file by copying it as a new file named `another_file`
5. Delete the first file `new_file`
6. Also delete the directory (you are currently in) `~/new_dir`. Does it work?

2.3 View Files

1. Which tools can you use to see the first/last lines of the file `~/exercises/P12931.txt`?
2. How to only show the first/last three lines (of the same file)?
3. How do you print the whole file on the screen?

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

2.5 Misc. terminal

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

2.6 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` and name it 'mydate'.
6. Check the permissions of the copied program 'mydate'
7. Change the permissions on 'mydate' to remove the executable permissions.
8. Check the permissions of the program 'mydate'
9. Change the permissions back so that the file is executable.
10. Try running it as `./mydate` or `adir/mydate` (depending on your current working directory)
11. Copy a textfile from a previous exercise into `adir`, then change the permissions, so you are not allowed to write to it. Test that you are still able to read the file via *cat*.
12. Then change the permissions so you can't read/cat it either. Test this by trying to read it via *cat*.
13. Change your working directory to `testpermissions`, and then try changing the permissions on the directory `adir` to non-executable.
14. What are the minimum permissions (on the directory) necessary for you to be able to execute `adir/mydate`?

2.7 Remote access

1. Login to machine "submaster.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:
5. Copy the file `/etc/motd` from machine submaster.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

2.8 IO and Redirections

1. Use date in conjunction 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* (page 22) symbol (|) and *sort* to sort this output *numerically*

2.9 Putting it all together

1. Create a new directory named `myscripts` in your homedirectory
2. Create an empty file named `mydate` in the newly created directory
3. Add the directory `~/myscripts` to your `PATH` environment variable
4. Use `echo` in combination with Redirection/Append to write "date" into the file `~/myscripts/mydate`

5. Change the permissions of the file `mydate` to be executable by you (and you only)
6. Run the file `mydate` (it should print the current date & time). Make sure you can run it from any directory (change to your homedirectory and just type `mydate`).

2.10 Bioinformatics

Let's do some bioinformatics analysis! You can find the famous BLAST tool installed at */g/software/bin/blastp*.

1. Typing the full path is too cumbersome, so let's append */g/software/bin* to your $PATH variable and ensure that it works by calling *blastp*.
2. When you run *blastp -help*, you notice that it has a lot of options! Use redirections in conjunction with grep to find out which options you need to specify a *input_file* and *database_name*.
3. Now run blastp using the following values as options:

   database_name = */g/data/ncbi-blast/db/swissprot*

   input_file = *suspect1.fasta*
4. Use either `less` or redirection to a file to manage the amount of information that `blastp` prints on your screen.

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]
- BioPieces [6] 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 [7]
- Useful bash one-liners for bioinformatics [8]
- Interactive explanation of your commandline: Explain Shell [9]
- Bash One-Liners Explained, Part III: All about redirections [10]
- Bash Redirections Cheat Sheet [11]
- Redirection Tutorial [12]

[1] http://linuxcommand.org/
[2] http://www.mn.uio.no/astro/english/services/it/help/basic-services/linux/guide.html
[3] http://www.oreilly.com/openbook/debian/book/ch04_01.html
[4] http://www.catcode.com/teachmod/
[5] http://www.cheat-sheets.org/#Linux
[6] http://code.google.com/p/biopieces
[7] https://code.google.com/p/google-styleguide
[8] https://github.com/stephenturner/oneliners
[9] http://www.explainshell.com
[10] http://www.catonmat.net/blog/bash-one-liners-explained-part-three
[11] http://www.catonmat.net/blog/bash-redirections-cheat-sheet
[12] http://wiki.bash-hackers.org/howto/redirection_tutorial

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 Recommended Reading: 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.

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

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 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).

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

Chapter 4

About Bio-IT

The Bio-IT Project aims to develop and strengthen the bioinformatics community at EMBL Heidelberg. It is made up of members across a range of disciplines in computational biology, in different Units and Core Facilities. The project aims to improve the standard of computational biology practised at EMBL Heidelberg, to encourage collaborations, and to provide a forum for discussion of issues and ideas relevant to bioinformatics here. The activities of the project include:: - the organisation and delivery of training courses such as this one - the provision of one-to-one training and consultancy - the organisation of social and networking events for the computational biology community - regular meetings to discuss issues and ideas - the development and maintenance of the Bio-IT Portal <http://bio-it.embl.de>

The Portal hosts information regarding upcoming courses and conferences/other events relevant to computational biology, resources to help with your work, and profiles of people involved in bioinformatics at EMBL. It is accessible from within the EMBL network (you must connect via VPN for off-site access).

4.1 Centres

EMBL Centres are 'horizontal', cross-departmental structures that promote innovative research projects across disciplines. All the EMBL Centres listed below have a strong computational component.

4.1.1 Biomolecular Network Analysis

The CBNA disseminates expertise, know-how and guidance in network integration and analysis throughout EMBL.

4.1.2 Statistical Data Analysis

The CSDA helps EMBL scientists to use adequate statistical methods for their specific technological or biological applications.

4.1.3 Modeling

The Centre for Biological Modeling (CBM) aims to support people to adopt mathematical modeling techniques into their everyday research.

Chapter 5

Acknowledgements

Handouts provided by EMBL Heidelberg Photolab (Many thanks to Udo Ringeisen)

EMBL Logo © EMBL Heidelberg

Graphic of the *Linux Filesystem* (page 2) taken from the SuSE 9.2 manual © Novell Inc.

Special thanks go to contributors / helping hands (alphabetical order):

- Christian Arnold
- Jean-Karim Hériché
- Yan Ping Yuan
- Bora Uyar
- Thomas Zichner

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/grep`

```
$ file /usr/bin/grep
/usr/bin/grep: 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 Copying / Deleting Files & Folders

1. Navigate to your home directory

```
$ cd ~
```

   or just

```
$ cd
```

2. In your homedirectory, create a new directory named `new_dir`

```
$  mkdir ~/new_dir
```

3. Change into this directory, create a new empty file in there named `new_file`, and make sure that the file was created:

```
$ cd ~/new_dir
$ touch new_file
$ ls new_file
```

4. Duplicate this file by copying it as a new file named `another_file`:

```
$ cp new_file another_file
```

5. Delete the first file `new_file`:

```
$ rm new_file
```

6. Also delete the directory (you are currently in) `~/new_dir`.

```
$ rmdir ~/new_dir
```

7. Did the deletion work? If not, try to remove all files from the directory first...:

```
$ rm ~/new_dir/*
$ rmdir ~/new_dir
```

6.3 View Files

1. Which tools can you use to see the first/last lines of the file `~/exercises/P12931.txt`?:

```
$ head ~/exercises/P12931.txt
$ tail ~/exercises/P12931.txt
```

2. How to only show the first/last three lines (of the same file)?:

```
$ head -n 3 ~/exercises/P12931.txt
$ tail -n 3 ~/exercises/P12931.txt
```

3. How do you print the whole file on the screen?:

```
$ cat ~/exercises/P12931.txt
```

or

```
$ less ~/exercises/P12931.txt
```

6.4 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 named `date` in the `/bin` directory

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

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

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

6.5 Misc. terminal

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

   ```
   $  clear
   ```

   or:

   ```
   $  reset
   ```

6.6 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` and name it 'mydate'.:

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

6. Check the permissions of the copied program 'mydate'

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

7. Change the permissions on 'mydate' to remove the executable permissions.:

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

8. Check the permissions of the program 'mydate'

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

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

```
$ adir/mydate
permission denied
```

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

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

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

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

12. Copy a textfile from a previous exercise into `adir`, then change the permissions, so you are not allowed to write to it. Test that you are still able to read the file via *cat*

```
$ 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. Test this by trying to read it via *cat*.

```
$ 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.

```
$ # no need to change directory,
$ # as 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/mydate`?

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

6.7 Remote access

1. Login to machine “submaster.embl.de” (using your own username)

```
$ ssh submaster.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 submaster.embl.de into your local home directory (using scp):

```
$ scp submaster.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
```

6.8 IO and Redirections

1. Use date in conjunction 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

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

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

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

```
$ cat ~/current_date
```

3. 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
```

4. (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
```

5. 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* (page 22) symbol (|) and *sort* to sort this output *numerically*:

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

6.9 Putting it all together

1. Create a new directory named `myscripts` in your homedirectory:

```
$ mkdir ~/myscripts
```

2. Create an empty file named `mydate` in the newly created directory:

```
$ touch ~/myscripts/mydate
```

3. Add the directory `~/myscripts` to your `PATH` environment variable:

```
$ export PATH=$PATH:~/myscripts
```

4. Use `echo` in combination with Redirection/Append to write "date" into the file `~/myscripts/mydate`:

```
$ echo "date" >> ~/myscripts/mydate
```

5. Change the permissions of the file `mydate` to be executable by you (and you only):

```
$ chmod u+x ~/myscripts/mydate
$ chmod go-x ~/myscripts/mydate
```

6. Run the file `mydate` (it should print the current date & time). Make sure you can run it from any directory (change to your homedirectory and just type `mydate`).:

```
$ mydate
```

Congratulation, you've just created and run your first shell script!

6.10 Bioinformatics

Let's do some bioinformatics analysis! You can find the famous BLAST tool installed at */g/software/bin/blastp*.

1. Typing the full path is too cumbersome, so let's append */g/software/bin* to your $PATH variable and ensure that it works by calling *blastp*.

```
$ export PATH=$PATH:/g/software/bin
$ blastp
```

2. When you run *blastp -help*, you notice that it has a lot of options! Use redirections in conjunction with grep to find out which options you need to specify a *input_file* and *database_name*.

```
$ blastp -help | grep input_file
[-subject subject_input_file] [-subject_loc range] [-query input_file]

$ blastp -help | grep database_name
search_strategy filename] [-task task_name] [-db database_name]
```

3. Now run blastp using the following values as options:

   database_name = /g/data/ncbi-blast/db/swissprot

   input_file = suspect1.fasta

```
$ blastp -db /g/data/ncbi-blast/db/swissprot -query suspect1 fasta
```

4. Use either `less` or a redirection into a file to manage the amount of information that `blastp` prints on your screen.:

```
$ blastp -db /g/data/ncbi-blast/db/swissprot -query suspect1 fasta | less
```

   or:

```
$ blastp -db /g/data/ncbi-blast/db/swissprot -query suspect1 fasta > blast_output
```

Index

W