# Lecture01: An Introduction to Unix

Who has done Unix before?!

## http://129.215.170.35/BPSM.html

Today we are going to "refreshen" our knowledge of Unix commands, and then use them to extract bits of information from a text file.

I have tried to make this guide as generally valid as possible, but there are many different flavours/versions of Unix available, so if you should find a command option behaving differently on your local machine you should consult the on-line manual page for that command (we'll see how later).

Most of the commands have numerous additional options that I have not mentioned, so for fuller information on these commands use the relevant on-line manual page.

The names of commands are printed in bold, and the names of

objects operated on by these commands (e.g. files, directories) are
printed in teletype.

# Index of Commands

awk - pattern scanning and processing language

cat, zcat - display or concatenate files

cd - change directory

chmod - change the permissions on a file or directory

cp - copy a file

cut - extract sections from each line of input

date - display the current date and

mkdir - make a directory

more, zmore - scan through a text file page by page

mv - move or rename files or directories

passwd - change your password

paste - join files horizontally

ps - list processes

time

find - find files of a specified name or type

ftp - file transfer program

grep, zgrep - searches files for a specified string or expression

gzip - compress a file

head, tail - display lines/characters from a file

kill - kill a process

ls - list names of files in a directory

man - display an on-line manual page

pwd - display the name of your current directory

rm - remove files or directories

rmdir - remove a directory

sort - sort and collate lines

ssh, scp - secure remote access

tar - create and use archives of files

uniq - unique

wc - counting words, letters, lines of files

## awk - Pattern scanning and processing language

A whole language in itself, invoking **`awk`** can very useful as part of a pipe (series of linked commands). **`awk`** can filter rows based on column criteria or pattern matching, use arrays, functions, etc. etc..

In `awk`, the notation `$1` means the contents of the first field, `$2` means the second field, etc.

For awk, the default field separator `FS` is a **space**, but we can tell it to use something else if we want.

```
The prickly hedgehog was hibernating.
 $1   $2         $3   $4     $5       (fields)

The prickly hedgehog was hibernating.
$1    $2 $3   $4 $5        (fields)
```

Text tables are frequently tab-delimited...

`awk '($3 == 5)' file1`

output rows of `file1` where the third field/column has a value equal to 5; this assumes that the field separator is a **space**. This command acts as a filtering step, and in these instances, the evaluation command is surrounded by ordinary `(` and `)` brackets inside the quotes.

`awk '{FS="\t"; if($3 == 5){print $0;}}' file1`

first, we have told awk that the field separator is a **tab**, which is written as "\t"; awk will then output rows ($0 means the whole line) where the third column of `file1` has a value equal to 5.

This command is evaluating, and then doing something based on the evaluation, and thus is an action command. As this is an action, these commands have `{` and `}` curly brackets inside the quotes.

```
awk '{FS="\t"; if($3 == "foobar1"){print $0;}}' file1
```

first, we have told awk that the field separator (FS) is a **tab**; awk will then output rows where the third column of `file1` is the text string "foobar1" (but not if it is foobar or foobar2)

```
awk '{FS="\t"; OFS="_"; if($3 == "foobar1"){print
$1,$2,$3;}}' file1
```

first, we have told awk that the field separator is a **tab**; we have also told awk that the output field separator (OFS) should be an underscore.
awk will then output the first, second and third fields ($1,$2,$3) of rows/lines of the file where the third column of `file1` is the text string is exactly "foobar1"; any output fields will be separated by underscores.

```
If file1 was these three lines of tab-delimited text:
F1 AS top one foobar1 X0
F2 AS mid foobar12 Y9
F3 AS bottom sample from the dataset foobar H3
```

```
the output would be
```

`F1_AS top one_foobar1`

---

```
awk '{FS="_"; print NF;}' file1
```

for each line of `file1`, using underscore "_" as field separator, output the **number** of fields on the line.

```
awk '{FS=" "; print "There are",NF,"fields on this line.";}'
file1
```

for each line of `file1`, using space " " as field separator, output a short sentence indicating the **number** of fields on the line.

```
awk '{FS="XXX"; print "The last field on this line is",$NF
;}' file1
```

for each line of `file1`, using "XXX" as field separator, output a short sentence that tells us what the **actual** last field on each line was.

---

```
awk '{FS="\t"; if($3 == "United Kingdom"){print $0;}}' file1
> file2
```

For each line, tell awk that field separator is tab. Determine if the third field/column of `file1` is "United Kingdom" and if so, save the whole line to `file2`.

```
awk '{FS=" "; if($3 == "United Kingdom"){print $0;}}' file1 >
file2
```

Field separator is space. Determine if the third field/column of `file1` is "United Kingdom" and if so, save the whole line to `file2`. We have specified that the field separator is a space. This makes no sense as "United Kingdom" would be two fields, as the separator is a space! You will **NOT** get the same answer as in the previous code line!

When awk code gets long, it is best to put it on several lines. Awk doesnt require you to do this, but it makes it easier to spot coding errors! And PDFs are much better....!

```
## My first short awk script
awk 'BEGIN{FS="\t";}
{
if(substr($3,1,3) == "Uni")

   {print $1;}
}' file1 >> file2
```

A brief comment line

Field separator is tab (telling awk only once).

Determine if the first three characters (a sub-string) of the third field/column (separated by tabs) of `file1` are "Uni" and if so, save the first field onto the end of `file2`.

Curly brackets `{` `}` for the action, ordinary `(` `)` brackets for the evaluation.

## cat, zcat - display or concatenate files

`cat` takes content of a file and sends it to the standard output (i.e. screen, unless redirected elsewhere)
Generally used either to read files, or to string together copies of several files, writing the output to a new file.

`cat ex1`
: displays the contents of the file `ex1`

`zcat ex1.gz`
: works the same as `cat`, but takes a gzipped file as input, then uncompresses "on the fly" to the screen, file still compressed.

`cat ex1 ex2 > newex`
: creates a new file `newex` containing copies of `ex1` and `ex2`, with the contents of `ex2` following the contents of `ex1`

`cat ex3 >> newex`
: adds (appends) the contents of `ex3` to the contents of `newex`, thus contents of `newex` are now `ex1` `ex2` `ex3`

## pwd - display the name of your current directory (folder)
## cd - change directory (folder)

The command `pwd` gives the full pathname of your current directory.
`cd` is used to change from one directory to another.

`cd dir1`
: changes directory so that `dir1` is your new current directory. `dir1` may be either the full pathname of the directory, or its pathname relative to the current directory.

`cd`
: changes directory to your home directory.

`cd -`
: changes directory to the previous directory you were in.

`cd ..`
: changes directory to the parent directory of your current directory,

i.e up one level.

```
cd ../..
```

changes directory to the "grand-parent" directory of your current directory, i.e up two levels.

## chmod - change the permissions on a file or directory

A directory listing generated using `ls -al` might look like this:

```
-rw-rw-r-- 1 someuser  somegroup  2015 Oct 14  2020 dna.txt
-rw-rw-r-- 1 someuser  somegroup    28 Oct  2  2020 exons.txt
-rw-rw-r-- 1 someuser  somegroup   485 Oct  2  2020 genomic_dna1.txt
-rw-rw-r-- 1 someuser2 admin       124 Sep 30  2020 plasmid_dna1.text
-rw-rw-r-- 1 someuser2 admin       124 Sep 30  2020 plasmid_dna1.foo
```

`chmod` enables us to alter the permissions on files and directories using either symbolic or octal numeric codes. Note that you can only change these values on files that you own!

The symbolic codes are:

| | | | | | |
|---|---|---|---|---|---|
| **u** | user | **+** | to add a permission | **r** | read |
| **g** | group | **-** | to remove a permission | **w** | write |
| **o** | other | **=** | to assign a permission explicitly | **x** | execute (for files), access (for directories) |

The following examples illustrate how these codes are used.

**`chmod u=rw`** `dna.txt`

sets the permissions on the file `dna.txt` to give the user read and write permission on `dna.txt`. No other permissions are altered.

**`chmod u+x,g+w,o-r`** `dna.txt`

alters the permissions on the file `dna.txt` to give the user execute permission on `dna.txt`, to give members of the user's group write permission on the file, and prevent any users not in this group from reading it.

**`chmod u+w,go-x`** `dir1`

gives the user write permission in the directory `dir1`, and prevents all other users having access to that directory (by using **`cd`**. They can still list its contents using `ls`, however.)

Files have permission values in octal format too, namely:

```
4 read
2 write
1 execute
```

so we can associate combinations (sums) of these values to "user", "group" and "other/world" categories to control what can and can't be done with a file.

`chmod 741 dna.txt`
: changes the permissions on `dna.txt` to be rwx for user (4+2+1), r-- (read-only, can't write or run) for the group, and --x (executable only, can't read or write) for the world

```
-rwx r-- --x
```

`chmod 750 dna.txt`
: changes the permissions on `dna.txt` to be rwx for user (4+2+1), r-x (read-only, but executable, 4+1) for the group, and --- (no permissions at all) for the world

```
-rwx r-x ---
```

## cp - copy a file

The command `cp` is used to make copies of files and directories (folders). Remember, each and every file has:

1. a **path**, default is "where you are right now on the filesystem"
2. a **name**, there is no default here!

When copying a file or directory, there has to be a source (where from?) and a destination (where to?)!

`cp file1 file2`

copies the contents of the file `file1` into a new file called `file2`. We havent given a path for either `file1` or `file2`, so the assumption of "where I am now" is made for both by default. Note that `cp` cannot copy a file onto itself.

`cp file3 file4 dir1`

creates copies (with the same names) of `file3` and `file4` in the directory `dir1`.
We haven't given a path for either `file3` or `file4`, so the assumption of "where I am now" is made for both by default.
`dir1` must already exist for the copying to succeed.

`cp dir1/file3 .`
: copies `file3` in `dir1` to where you are, i.e the dot means "present working directory".

`cp dir1/file3 dir2/file4`
: copies `file3` in `dir1` to `file4` in `dir2`.

`cp -u file3 dir1`
: if source `file3` is newer than the destination `file3` or when the destination `file3` is missing, updates copy of `file3` within the directory `dir1`. `dir1` must already exist for the copying to succeed.

`cp -r dir2 dir3`
: recursively copies the directory `dir2`, together with its contents and subdirectories, to the directory `dir3`.
If `dir3` does not exist, it is created by `cp`, and the contents and subdirectories of `dir2` are recreated within it.
If `dir3` does exist, a subdirectory called `dir2` is created within it, containing a copy of all the contents of the original `dir2`.

## cut - extract sections from each line of input

The command `cut` is used to extract sections from each line of input; this can be done at the level of bytes (-b), characters (-c), or fields (-f) separated by a delimiter (-d, the tab character by default).

`cut -c 3-12 file1`
: output characters 3 to 12 of each line of `file1`.

`cut -f 3,5,9 file1`
: output fields 3, 5 and 9 of each line of `file1`. Assumes tab delimiter.

`cut -d "_" -f 4- file1`
: output the fourth and upwards "_" separated field of each line of `file1`

If `file1` contained this single line of text:

```
this_is_not_a_silly_sentence
```

the output of `cut -d "_" -f 4- file1` would be:

```
a_silly_sentence
```

the output of `cut -d "_" -f 5 file1` would be:

```
silly
```

## date - display the current date and time

`date`

returns information on the current date and time in the format shown below

```
Mon Jul 18 11:23:56 BST 2016
```

`date -u`

returns information on the current date and time in Coordinated Universal Time (UTC) format (GMT is a time **zone** and UTC is a time **standard**.)

```
Mon Jul 18 10:23:56 UTC 2016
```

The official abbreviation for Coordinated Universal Time is UTC. It came about in 1967 as a compromise between English and French speakers.

- In English,Coordinated Universal Time, abbreviated CUT.
- En français, Temps Universel Coordonné, abbreviated TUC.

The International Telecommunication Union (ITU) and the International Astronomical Union (IAU) designated one single abbreviation for use in all languages: UTC.

https://www.timeanddate.com/time/aboututc.html

It is possible to alter the format of the output from date. For example, using the command line

```
date '+The date is %d/%m/%y, and the time is %H:%M:%S.'
```

at exactly 11.30am on 30th January 2020, would produce the output

```
The date is 30/01/20, and the time is 11:30:00.
```

To show the time in seconds since 1970-01-01 (Unix epoch time):

```
date +%s
```

returns a number like `1505483716`

## find - find files of a specified name or type

`find` searches for files in a named directory and all its subdirectories.

```
find . -name '*.f'
```

searches the current directory and all its subdirectories for files ending in `.f`, and writes their names to the standard output. In some versions of Unix the names of the files will only be written out if the `-print` option is used.

```
find /local -name core -user user1
```

searches the directory `/local` and its subdirectories for files called `core` belonging to the user `user1` and writes their full file names to the standard output.

## ftp - file transfer program

`ftp` is an interactive file transfer program. While logged on to one system (described as the local system), `ftp` is used to log on to another system (described as the remote system) that files are to be transferred to or from. As well as file transfers, it allows the inspection of directory contents on the remote system. There are numerous options and commands associated with `ftp`, and `man ftp` will give details of those.

**WARNING!** When you use `ftp` the communications between the systems are not encrypted. This means that your password could be snooped if you use it make an `ftp` connection. If you wish to transfer files between two systems where you have accounts it is better to use the commands `sftp` (secure file transfer program) or `scp` (secure remote file copy program) if available, as they provide encrypted file transfer. See the section on `ssh` for examples.

Some systems offer a service called "anonymous ftp", usually to allow general access to certain archives. To use such a service, enter `anonymous` instead of your username when you ftp to the system. It is fairly standard practice for the remote system to ask you to give your email address in place of a password. Once you have logged on you will have read access in a limited set of directories, usually within the /pub directory tree.

---

```
ftp -i ftp.ncbi.nlm.nih.gov
```

If the connection to the remote system ftp.ncbi.nlm.nih.gov is established, it will respond:

```
Connected to ftp.ncbi.nlm.nih.gov.
220-
 This warning banner provides privacy and security notices consistent with
 applicable federal laws, directives, and other federal guidance for accessing
 this Government system, which includes all devices/storage media attached to
 this system. This system is provided for Government-authorized use only.
 Unauthorized or improper use of this system is prohibited and may result in
 disciplinary action and/or civil and criminal penalties. At any time, and for
 any lawful Government purpose, the government may monitor, record, and audit
 your system usage and/or intercept, search and seize any communication or data
 transiting or stored on this system. Therefore, you have no reasonable
 expectation of privacy. Any communication or data transiting or stored on this
 system may be disclosed or used for any lawful Government purpose.
220 FTP Server ready.
Name (ftp.ncbi.nlm.nih.gov:user1):
```

(supposing `user1` is your username on your local system). Enter `anonymous` and press `Return`. You will then be asked to enter your email address instead of a password.

After logging in, some Unix commands, such as **cd** and **ls**, will be available. Other useful commands are:

`help`
: lists the commands available to you while using **ftp**

`get remote1 local1`
: creates a copy on your local system of the file `remote1` from the remote system. On your local system this new file will be called `local1`. If no name is specified for the file on the local system, it will be given the same name as the file on the remote system.

`mget remote1*`
: retrieves all files starting with the name `remote1` from the remote system and puts them in your local working directory.

`quit`
: finishes the **ftp** session. **bye** and **close** can also be used to do

this.

## grep, zgrep - searches files for a specified string or expression

`grep` searches for lines containing a specified pattern and, by default, writes them to the standard output.
`zgrep` does the same thing as `grep`, but takes a gzipped file as input.
**BOTH are case sensitive!**

`grep "motif1" file1`
  searches the file `file1` for lines containing the pattern `"motif1"`.

`grep "^motif1" file1`
  searches the file `file1` for lines containing the pattern `"motif1"` at the beginning of a word/string.

`grep "motif1$" file1`
searches the file `file1` for lines containing the pattern `"motif1"` at the end of a word/string.

`grep -w "motif1" file1`
searches the file `file1` for lines containing the word `"motif1"`.

`cat file1 | grep "motif1"`

: will apply `grep` to the standard input/output via a pipe, looking for lines containing `motif1`.

`grep "motif1" file1 file2 filen`

: will search the files `file1`, `file2` and `filen` for the pattern `motif1`.

`grep "motif1" a*`

: will search all the files in the current directory with names beginning with 'a' for the pattern `motif1`. In this instance, the `*` is a wild-card that means "anything".

`grep -c "motif1" file1`

: will count the number of lines containing `motif1` instead of outputting the lines themselves.

`grep -m3 "motif1" file1`

will output the first three lines that contain `motif1`.

`grep -v "motif1" file1`
will write out the lines of `file1` that do NOT contain `motif1`.

`grep -E "motif1|motif2" file1`
searches the file `file1` for lines containing one or more of the two patterns `motif1` and `motif2`.

`grep -Ei "motif1|motif2" file1`
searches, in a case-insensitive mannner, the file `file1` for lines containing one or more of the two patterns `motif1` and `motif2`.

`cat file1 | grep "motif1"`
will apply `grep` to the standard input/output via a pipe, looking for lines containing `motif1`.

`cat file1 | grep "motif1" | grep "motif3"`
will apply `grep` to the standard input/output via a pipe, looking for lines containing `motif1` and then within those, lines that contain

`motif3`. The lines output will thus have `motif1` AND `motif3`.

**`grep --help`**

will tell you about the huge number of options available.

## gzip, pigz - compress a file

`gzip` reduces the size of named files, replacing them with files of the same name extended by `.gz` . The amount of space saved by compression varies. `pigz` is a similar utility that is able to work over multiple threads, and thus can be substantially faster.

`gzip file1`
: results in a compressed file called `file1.gz`, and deletes `file1`.

`gzip -v file2`
: compresses `file2` and gives information, in the format shown below, on the percentage of the file's size that has been saved by compression:-

```
file2 : Compression 50.26 -- replaced with file2.gz
```

To restore files to their original state use the command `gunzip` or `gzip -d`.

```
gzip -d file2.gz
```

will replace `file2.gz` with the uncompressed file `file2`.

## head, tail - display lines of a file

`head -n5 file1`
: displays the first 5 lines of `file1`.

`tail -n5 file1`
: displays the last 5 lines of `file1`.

`tail -n +2 file1`
: shows all lines of `file1` from the second line onwards.

`head -n5 file1*`
: displays the first 5 lines of all files whose names start with file1.

`tail -n5 file1*`
: displays the last 5 lines of all files whose names start with file1.

```
tail --silent -c5 file1*
```

outputs the last 5 characters of all files whose names start with file1, silently suppressing the filenames. Notice that the count includes the newline character at the end of each line, so you'll get 4 characters, visible or otherwise, back.

## kill - kill a process

To kill a process using `kill` requires knowing the process id (PID).
This can be found by using the command **ps**:

```
ps aux | grep "myoutofcontrolprocess"
```

```
user      4724  0.0  0.0  39904  4028 pts/3    R+   Sep07   0:14 myoutofcontrolprocess
```

will identify the process.

```
kill STOP 4724
```

will stop (but not remove, i.e. pause) the process.

```
kill CONT 4724
```

will continue the STOPped process.

```
kill 4724
```

should kill the process.

```
kill -9 4724
```

will kill the process, and all its dependants.

## ls - list names of files in a directory

`ls` lists the contents of a directory, and can be used to obtain information on the files and directories within it.

`ls`

  lists the contents of the current directory in multiple columns across the screen, alphabetically, case sensitive.

`ls -1`

  lists the contents of the current directory in a **single** column on the screen.

`ls dir1`

  lists the names of the files and directories in the directory `dir1`, (excluding files whose names begin with . ).

`ls -R dir1`

also lists the contents of any subdirectories that `dir1` contains.

`ls -a dir1`
: will list all contents of `dir1`, (including files whose names begin with . ).

`ls -l file1`
: gives long details of the access permissions for the file `file1`, its size in kbytes, and the time it was last altered.

`ls -l dir1`
: gives long format information on the contents of the directory `dir1`.

---

`ls -al`
: gives full information on the contents of the current directory.

`ls -alrt`
: gives, in reverse time order, full information on the contents of the current directory.

`ls -ld dir1`
: To obtain the information on `dir1` itself, rather than its contents.

## man - display an on-line manual page

`man` displays on-line reference manual pages.

`man command1`
: will display the manual page for `command1`, e.g `man cp`, `man man`.

`man -k keyword`
: lists the manual page subjects that have keyword in their headings. This is useful if you do not yet know the name of a command you are seeking information about.

`man -Mpath command1`
: is used to change the set of directories that man searches for manual pages on `command1`

## mkdir - make a directory

`mkdir` is used to create new directories. In order to do this you must have write permission in the parent directory of the new directory.

`mkdir newdir`
: will make a new directory called `newdir`.

`mkdir -p dir1/dir2/newdir`
: will create newdir and its parent directories `dir1` and `dir2`, if these do not already exist.

## more, zmore - scan through a text file page by page

`more` displays the contents of a file on a terminal one screenful at a time.

`zmore` does the same as `more`, but takes a gzipped file as input.

`more file1`
: starts by displaying the beginning of `file1`. It will scroll up one line every time the return key is pressed, and one screenful every time the space bar is pressed. Type **?** for details of the commands available within `more`. Type **q** if you wish to quit more before the end of `file1` is reached.

`more -n file1`
: will cause `n` lines of `file1` to be displayed in each screenful instead of the default (which is two lines less than the number of

lines that will fit into the terminal's screen).

## mv - move or rename files or directories

`mv` is used to change the name of files or directories, or to move them into other directories.

`mv file1 file2`

  changes the name of a file from `file1` to `file2` unless `dir2` already exists, in which case `dir1` will be moved into `dir2`.

`mv dir1 dir2`

  changes the name of a directory from `dir1` to `dir2`.

`mv file1 file2 dir3`

  moves the files `file1` and `file2` into the directory `dir3`.

## passwd - change your password

Use `passwd` when you wish to change your password.

You will be prompted once for your current password, and twice for your new password.

Neither password will be displayed on the screen.

Use with care....!

## paste - join files horizontally

For each corresponding line, `paste` will append the contents of each file at that line to its output along with a tab. When it has completed its operation for the last file, paste will output a newline character and move on to the next line.

`paste file1 file2 > file3`
: line by line, paste the contents of `file2` next to the contents of `file1`, separated by a tab, and output to `file3`.

`paste -d az file1 file2 file3 > file123`
: line by line, paste the contents of `file1`, `file2` and `file3`, next to each other, alternately **d**elimited by "a" and then "z", and output the result to `file123`.

`paste -s file1 file2 > file3`
: output the contents of `file1` and `file2` serially as tab-separated

items in horizontal orientation to `file3`

```
paste file1 file2
file1line1 file2line1
file1line2 file2line2
file1line3 file2line3
```

```
paste -s file1 file2
file1line1 file1line2 file1line3
file2line1 file2line2 file2line3
```

## ps - list processes

`ps` displays information on processes currently running on your machine. This information includes the process id, the controlling terminal (if there is one), the cpu time used so far, and the name of the command being run.

**NOTE:** `ps` is a command whose options vary considerably in different versions of Unix. Use `man ps` for details of all the options available on the machine you are using.

`ps`
: gives brief details of your own processes in your current session.

`ps -au`
: gives fuller details of your own processes in your current session.

`ps -au user1`

gives fuller details of processes owned by `user1` in your current session.

`ps aux`

gives fuller details of all processes owned by all users. Note missing "-", but effect depends on what flavour of Unix...

## rm - remove files or directories: PERMANENTLY...

`rm` is used to remove files. In order to remove a file you must have write permission in its directory, but it is not necessary to have read or write permission on the file itself.

`rm file1`
: will delete the file `file1`, permanently, so use with care.

`rm -i file1`
: instead, you will be asked if you wish to delete `file1`, and the file will not be deleted unless you answer `y`. This is a useful safety check when deleting lots of files, and you should consider aliasing it to minimise unwanted unhappy events....

`rm -r dir1`

recursively deletes the contents of `dir1`, its subdirectories, and `dir1` itself, and should be used with caution.

**`rm -fr dir1`**

forcibly and recursively deletes the contents of `dir1`, its subdirectories, and `dir1` itself. This is a "nuclear" option, so be very afraid!

## rmdir - remove a directory

`rmdir` removes named empty directories. If you need to delete a non-empty directory `rm -r` can be used instead.

`rmdir exdir`
: will remove the empty directory `exdir`.

## `sort` - sort and collate lines

The command `sort` sorts and collates lines in files, sending the results to the standard output.

If no file names are given, `sort` acts on the standard input.

By default, `sort` sorts lines using a character by character comparison, working from left to right, and using the order of the ASCII character set.

`sort -d`
: uses "dictionary order", in which only letters, digits, and white-space characters are considered in the comparisons.

`sort -r`
: reverses the order of the collating sequence.

`sort -n`
: sorts lines according to the arithmetic value of leading numeric strings. Leading blanks are ignored when this option is used, (except in some System V versions of `sort`, which treat leading blanks as significant. To be certain of ignoring leading blanks use `sort -bn` instead).

```
sort -t$'\t' -k1,1 file1
```

sorts the file `file1` on the first column; default field delimiter is "non-blank to blank" transition so tab and space should work equally well; adding the `-t$'\t'` is a safer way to ensure tabs are used.
We write the column twice (e.g. -k1,1) to ensure that sort is only using the given column (in this case, column 1) when determining the "sort key order"; everything on each line can be output, but the order will determined by the -k1,1 sort key.

```
sort -k1,1 -k2,2nr file1
```

sorts the file `file1` on the first column, and then on the second column, treating the latter as numeric, outputting the lines in decreasing numeric value order of column 2.

```
cat file1 | grep "motif1" | sort -k1,1 -k2,2nr
```

....cats the file `file1`
....retains any line that has the string `"motif1"` in it,
....sorts on the first column, and then on the second column, treating the latter as numeric, outputting the lines in decreasing

numeric value of column 2.

## ssh - secure remote access

`ssh` (also known as `slogin`) is used for logging onto a remote system, and provides secure encrypted communications between the local and remote systems using the SSH protocol.

The remote system must be running an SSH server for such connections to be possible. You can ssh to a machine using its name, or, if you know it, its unique IP address. For example:

```
ssh bioinfmsc5.bio.ed.ac.uk
```

or

```
ssh 129.215.237.197
```

initiates a login connection to the course server bioinfmsc5.bio.ed.ac.uk, which has IP address 129.215.237.197.

```
ssh someuseruun@bioinfmsc5.bio.ed.ac.uk
```

initiates a login connection for `someuseruun` to the course server bioinfmsc5.bio.ed.ac.uk.

You will need to use your EASE login details (both username and password).

If you wish to transfer files over an encrypted connection you can use `sftp` (secure remote file transfer program) or `scp` (secure remote file copy program); authentication is handled as for `ssh`.

```
sftp bioinfmsc5.bio.ed.ac.uk
```

Once you have authenticated access to bioinfmsc5.bio.ed.ac.uk, you will be in your home directory. You can use the command `cd` to change directories on bioinfmsc5.bio.ed.ac.uk and `lcd` to change directories on your local system; `get` can be used to transfer files from the remote system, and `put` to transfer files to the remote system. The command `quit` will terminate the `sftp` session.

Alternatively, you could use `scp` to transfer files.

```
scp remoteserver:file1 newfile1
```

is used to transfer a copy of the file `file1` in your home directory on the remote server to the current directory on the local system, naming the file `newfile1`.

```
scp file2 remoteserver:newfile2
```

to copy the local file `file2` to the remote system, calling the copy `newfile2`

## tar - create and use archives of files

`tar` can be used to create and manage an archive of a set of files. Tar stands for TapeARchive.

`tar cf archive1.tar`
: creates an archive file called `archive1.tar` containing the contents of the current directory (and any subdirectories it contains). The `c` option stands for "create" and the `f` for "filename".

`tar cf archive1.tar *.html`
: creates an archive file called `archive1.tar` containing all html files in the current directory.

`tar cf archive2.tar mydir`

creates an archive file called `archive2.tar` containing the contents of the directory `mydir`.

**`tar tvf`** `archive1.tar`

lists the contents of the archive file `archive1.tar`. The **`t`** stands for "list" and the **`v`** for "verbose listing".

---

`tar xf archive1.tar`
: extracts the contents of `archive1.tar` and copy them into the current directory. The `x` stands for "extract".

`tar xf archive1.tar file2`
: extracts `file2` from `archive1.tar` (if `file2` is in the archive).

`tar uf archive1.tar file2`
: If `file2` is not already in the archive it will be added. The `u` stands for "update". If there is already a file called `file2` in the archive, `file2` will be appended to the archive if it has a more recent timestamp than the `file2` already in the archive. This means the most recent version of `file2` will be obtained when `file2` is extracted from the archive.

## uniq - unique

`uniq` is frequently used after or with the `sort` command to determine frequencies of lines/elements.

`uniq file1`
: listing of non-redundant lines in `file1`, but only if it is **sequential** lines that are not identical.

`cat file1 | sort | uniq -c`
: number of occurrences of each non-redundant line of `file1`.

`sort file1 | uniq -c`
: does the same thing, just a shorter command: count the number of occurrences of each non-redundant line of `file1`.

## wc - counting words, letters, lines of files

`wc file1`
: newline count, word count, and character/byte count of `file1`.

```
cat file1
```

```
file1line1
file1line2
file1line3
```

```
wc file1
```

```
 3   3 33 file1
```

`wc -c file1`
: number of characters in `file1`.

`wc -w file1`

number of words in `file1`.

`wc -l file1`
: number of lines in `file1`.

...and now let's try the exercises...