Tutorial

The power of a command line

Computing is an integral part of science and a great majority of powerful computational tools require use of command line. Interaction with most HPC resources also happens through the command line.

“Command-line interfaces are often preferred by more advanced computer users, as they often provide a more concise and powerful means to control a program or operating system.” Wikipedia

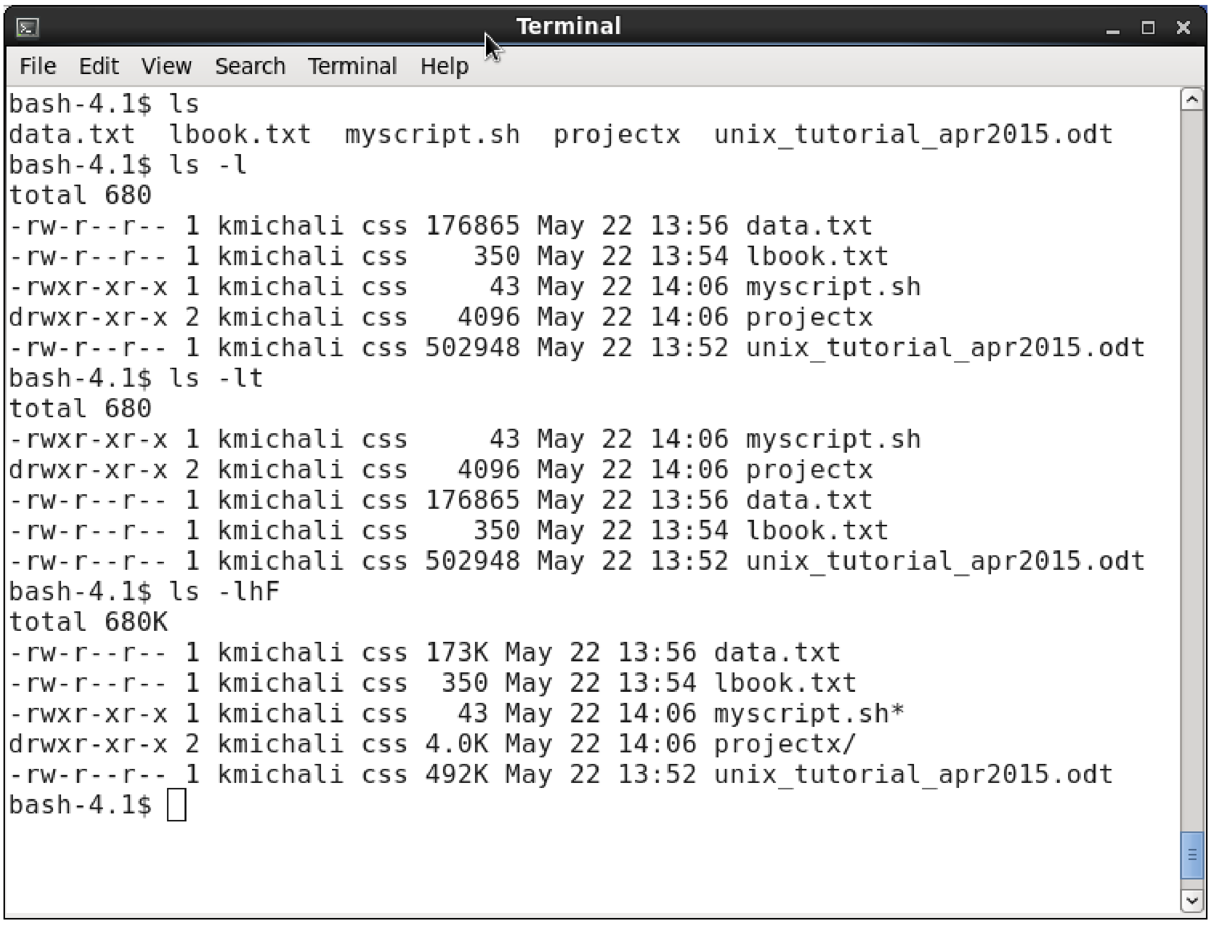
The user interface is text based, commands are typed on the line that begins with the prompt (for example a dollar sign) and executed by pressing enter.

Directory listing

The command ls (short for list) produces a simple list of files and directories residing in the current directory. The list does not contain any details such as size or modification date. To display a desired information or format, one can modify the behaviour of the command using various command options. For example, to get all file properties, one can use ls -l that produces a "long" format displaying file permission, ownership, size and data of modification. Other useful options are -t that orders directory listing by modification date (ls -lt) and -h that produces easy to read file size format. Using ls -lF adds slash "/" character to directories and star "*" to executables.

Most of command line tools come with multitude of options that can be displayed by typing "man command", e.g. man ls.

The figure below demonstrates the use of the ls command. The directory contains several files, one executable and one directory. The largest file is 492Kb.



```
bash-4.1$ ls
data.txt  lbook.txt  myscript.sh  projectx  unix_tutorial_apr2015.odt
bash-4.1$ ls -l
total 680
-rw-r--r-- 1 kmichali css 176865 May 22 13:56 data.txt
-rw-r--r-- 1 kmichali css    350 May 22 13:54 lbook.txt
-rwxr-xr-x 1 kmichali css     43 May 22 14:06 myscript.sh
drwxr-xr-x 2 kmichali css   4096 May 22 14:06 projectx
-rw-r--r-- 1 kmichali css 502948 May 22 13:52 unix_tutorial_apr2015.odt
bash-4.1$ ls -lt
total 680
-rwxr-xr-x 1 kmichali css     43 May 22 14:06 myscript.sh
drwxr-xr-x 2 kmichali css   4096 May 22 14:06 projectx
-rw-r--r-- 1 kmichali css 176865 May 22 13:56 data.txt
-rw-r--r-- 1 kmichali css    350 May 22 13:54 lbook.txt
-rw-r--r-- 1 kmichali css 502948 May 22 13:52 unix_tutorial_apr2015.odt
bash-4.1$ ls -lhF
total 680K
-rw-r--r-- 1 kmichali css 173K May 22 13:56 data.txt
-rw-r--r-- 1 kmichali css  350 May 22 13:54 lbook.txt
-rwxr-xr-x 1 kmichali css   43 May 22 14:06 myscript.sh*
drwxr-xr-x 2 kmichali css 4.0K May 22 14:06 projectx/
-rw-r--r-- 1 kmichali css 492K May 22 13:52 unix_tutorial_apr2015.odt
bash-4.1$ 
```

Note: On many systems, "ls -l" is aliased to "ll".

File ownership and permissions

The output of "ls –l" above produces extra information. From the left:

- permissions statement
- number of hard links (ignore for now)
- user name of the owner
- group that owner belongs to
- file size
- date of last modification
- file/directory name

File ownership and permissions provide control over actions performed on files and directories. In the above caption the owner is "kmichali" and the group is "css". The permission statement consists of 10 positions: -rwxrwxrwx. The first position is reserved for a directory sign (d). Ordinary files have just a "-" sign. The next three rwx triads specify permissions for the owner, group and everyone else. Files (and directories) can be read (r), written into (w) or executed (x). Each triad holds rwx permissions always in the same order.

For example, in the –rwsrwxrwx statement all permissions are set and all users can read, write into and execute a file. In the -rw------- statement, the file can only be read and written to by the owner.

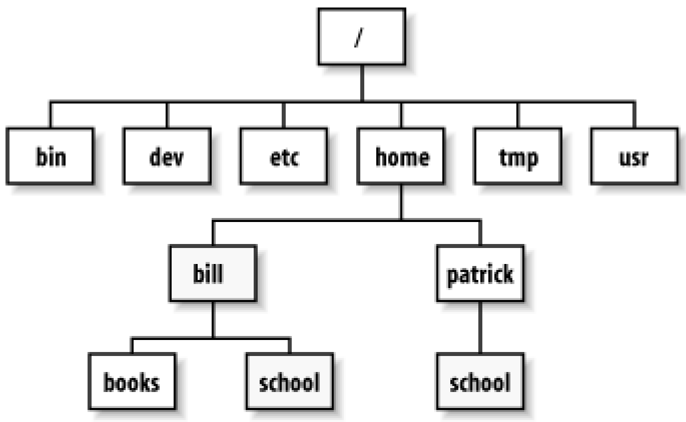
Path

The path specifies a position in the directory tree, it is a sequence of directory names separated by a forward slash. For example, when a user logs into a machine, they find themselves in their home directory and path looks like this "/home/someuser". Users have full permissions to all the files and directories in this space.

The path can be specified as absolute or relative.

- Absolute path always starts at at the top of the tree (at the root directory), for example "/home/username". The absolute path always starts with a slash "/" (root) that is the head of the filesystem.
- The relative path is the path to the destination from your current position and it does not start with a "/", e.g. "../home/otheruser" (the double dot means one directory level up).

The image below shows a small part of a hypothetical Linux directory tree. The root "/" usually contains subdirectories named bin, dev, etc, home, usr and tmp. User home directories are located in "/home". If you are Bill, your home directory is "/home/bill".   If your current project is in the directory "school", you have a choice of expressing the path as absolute "/home/bill/school" or as relative "./school" (where the dot stands for current directory). In this case, the relative path is convenient to use. However, if you wanted to share data with Patrick, the absolute path would be "/home/patrick/school" while relative "../../home/patrick/school".   Here, the absolute path is simpler to work with.



Changing directories

The command for changing a current directory is "cd" (change directory). The argument for cd is a path (absolute or relative). Consider the example above, as Bill you start your work in /home/bill. Typing "cd school" changes current working directory to "/home/bill/school". The command "pwd" or print working directory displays the current directory.

Special notations

There are few special notations to remember when navigating the directories.

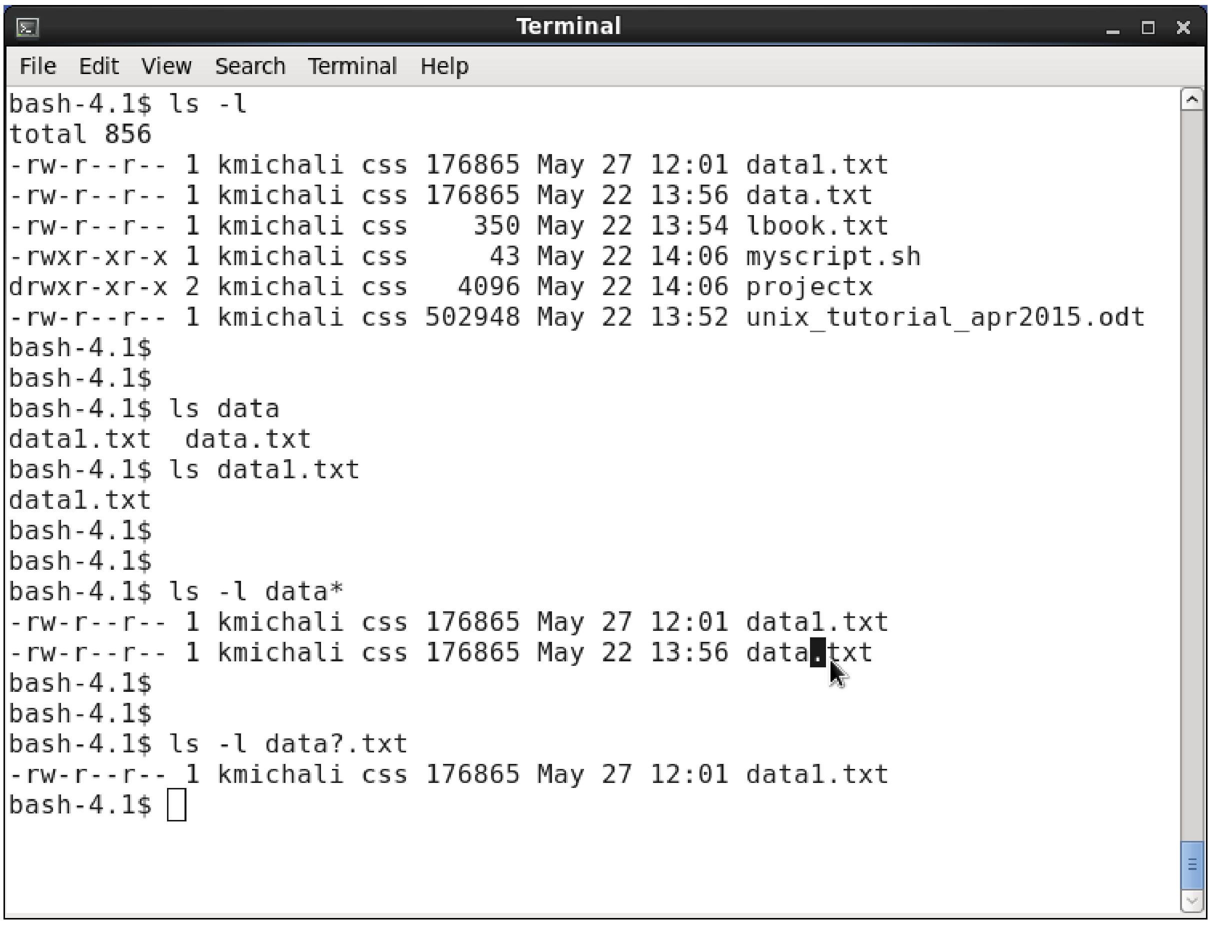
- Typing “cd ../” will take you one directory level up (from “/home/bill” to “/home”).
- The sign “./” means current directory (“work” is the same as “./work”).
- Typing “cd” anywhere will change the working directory to user's home directory.
- You can use “~” in the path to substitute for your home directory (e.g. “cd ~/school” is the same as “cd /home/bill/school”).
- Typing “cd -” will send you back where you just came from. E.g. you start at “/home/bill/school”, then “cd ../books”. When you need to go back to “/home/bill/school”, type “cd -”.

Autocompletion and wildcards

Autocompletion reduces typing and errors dramatically as it completes full file (or directory) names. Start by typing a file name and press the TAB key. If the fragment of the name is unique, pressing the TAB key will complete the name. If the fragment is not unique, typing a second TAB will produce list of matching filenames.

Wildcards * and ? are symbols that can be used in filename pattern matching. Often, wildcards are often used together with the ls command. For example, “ls results*” produces a list of items beginning with “results”. The wild card “*” matches any string of any length including an empty one and “?” matches exactly one character.

In the example below, the current directory contains files data.txt and data1.txt. If you type "ls -l data" and press TAB twice, the system returns both, data.txt and data1.txt as possible matches. If "1" is added, there is only one possible match and the file data1.txt is listed. The next command uses the wildcard "*" to list files. Both, data.txt and data1.txt are matched by "data*". The expression "data?.txt" matches only data1.txt since "?" represents exactly one character.



```
bash-4.1$ ls -l
total 856
-rw-r--r-- 1 kmichali css 176865 May 27 12:01 data1.txt
-rw-r--r-- 1 kmichali css 176865 May 22 13:56 data.txt
-rw-r--r-- 1 kmichali css    350 May 22 13:54 lbook.txt
-rwxr-xr-x 1 kmichali css     43 May 22 14:06 myscript.sh
drwxr-xr-x 2 kmichali css   4096 May 22 14:06 projectx
-rw-r--r-- 1 kmichali css 502948 May 22 13:52 unix_tutorial_apr2015.odt
bash-4.1$
bash-4.1$
bash-4.1$ ls data
data1.txt  data.txt
bash-4.1$ ls data1.txt
data1.txt
bash-4.1$
bash-4.1$
bash-4.1$ ls -l data*
-rw-r--r-- 1 kmichali css 176865 May 27 12:01 data1.txt
-rw-r--r-- 1 kmichali css 176865 May 22 13:56 data.txt
bash-4.1$
bash-4.1$
bash-4.1$ ls -l data?.txt
-rw-r--r-- 1 kmichali css 176865 May 27 12:01 data1.txt
bash-4.1$ 
```

Managing files and directories

This section contains examples of commands that are used to organize files and directories (or folders). Creating and deleting directories, copying, moving and deleting files are discussed.

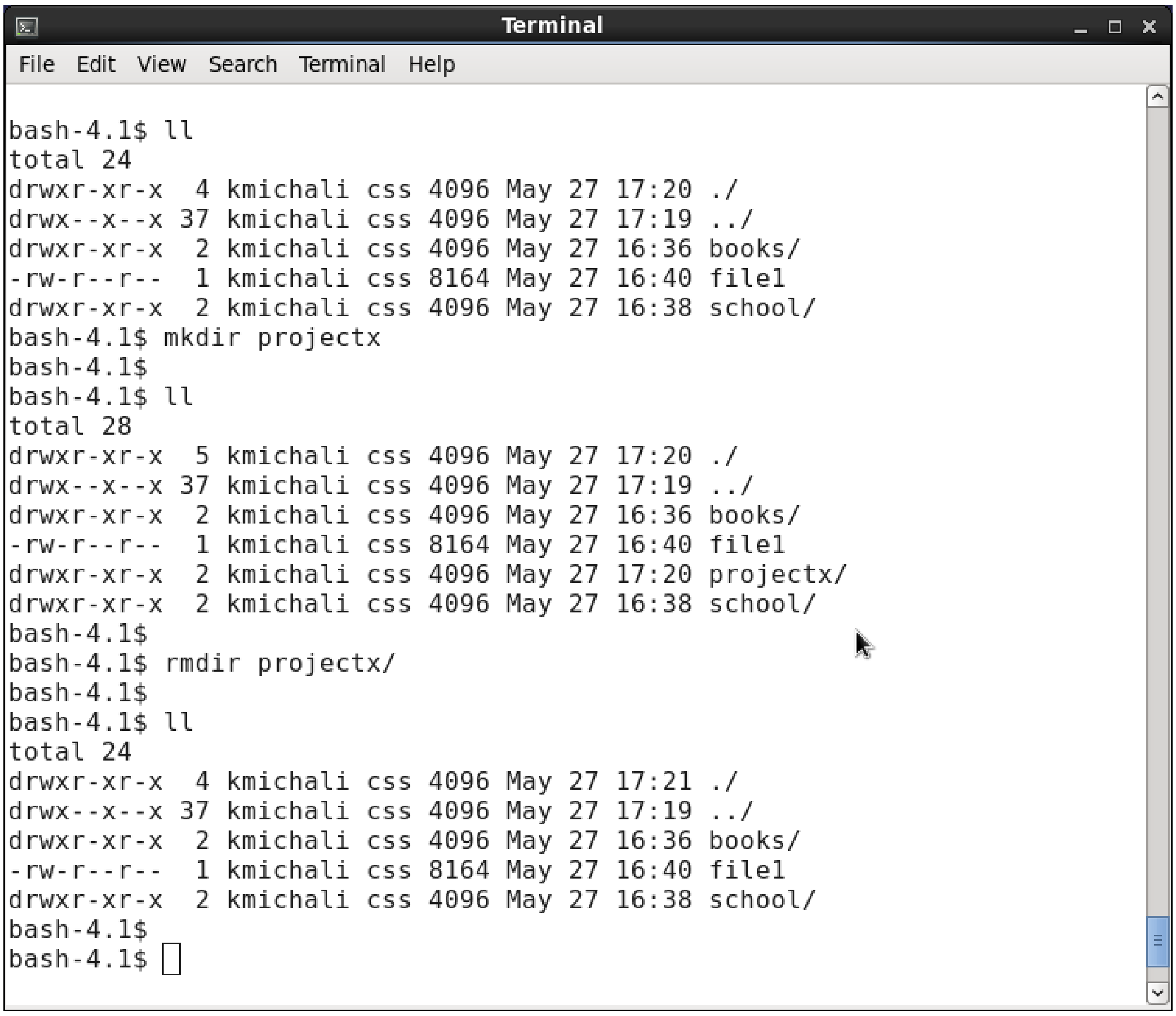
Make a directory – mkdir

The "mkdir" (make directory) command takes one argument, the path to the new directory. All the other components of the path must exist. For example, the "mkdir /home/bill/projectx" will create a directory called "projectx" in Bill's home directory (if /home/bill exists).   The same can be done by the following sequence of commands "cd /home/bill; mkdir projectx".

Remove a directory – rmdir

An empty directory can be removed using the rmdir (remove directory) command.   If the directory contains any files, they have to be removed first. The directory "projectx" from the above example can be removed with "rmdir /home/bill/projectx" or with "cd /home/bill; rmdir projectx".

The example below demonstrates the mkdir and rmdir commands. The directory "projectx" is created, checked with the "ll" command and subsequently deleted. Note that the command "rmdir projectx/" contains an optional slash.

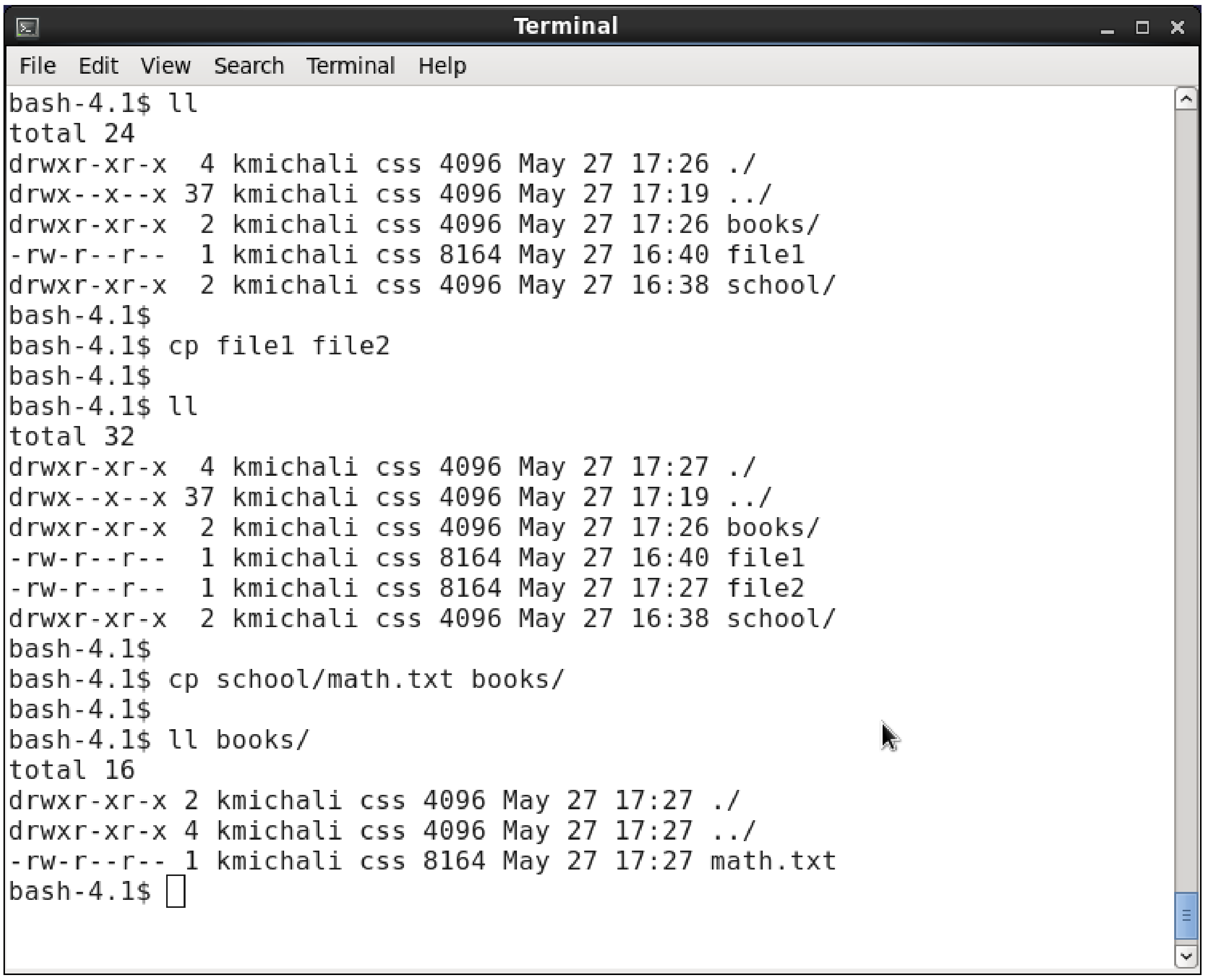


```
bash-4.1$ ll
total 24
drwxr-xr-x  4 kmichali css 4096 May 27 17:20 ./
drwx--x--x 37 kmichali css 4096 May 27 17:19 ../
drwxr-xr-x  2 kmichali css 4096 May 27 16:36 books/
-rw-r--r--  1 kmichali css 8164 May 27 16:40 file1
drwxr-xr-x  2 kmichali css 4096 May 27 16:38 school/
bash-4.1$ mkdir projectx
bash-4.1$
bash-4.1$ ll
total 28
drwxr-xr-x  5 kmichali css 4096 May 27 17:20 ./
drwx--x--x 37 kmichali css 4096 May 27 17:19 ../
drwxr-xr-x  2 kmichali css 4096 May 27 16:36 books/
-rw-r--r--  1 kmichali css 8164 May 27 16:40 file1
drwxr-xr-x  2 kmichali css 4096 May 27 17:20 projectx/
drwxr-xr-x  2 kmichali css 4096 May 27 16:38 school/
bash-4.1$
bash-4.1$ rmdir projectx/
bash-4.1$
bash-4.1$ ll
total 24
drwxr-xr-x  4 kmichali css 4096 May 27 17:21 ./
drwx--x--x 37 kmichali css 4096 May 27 17:19 ../
drwxr-xr-x  2 kmichali css 4096 May 27 16:36 books/
-rw-r--r--  1 kmichali css 8164 May 27 16:40 file1
drwxr-xr-x  2 kmichali css 4096 May 27 16:38 school/
bash-4.1$
bash-4.1$
```

Copy a file – cp

Files can be copied with the "cp" (copy) command. The command takes two arguments - a target path and a destination path separated by a space. For example, "cp file1 file2" makes a copy of the file1 in the current directory and names it file2. In a more complicated example, the command "cp /home/bill/school/math.txt /home/bill/books/" results in a copy of the file math.txt in the books folder; the file math.xls was both, copied and placed in a different directory at once. The copy command can be used to copy multiple files at once with help of wildcards; "cp /home/bill/books/algebra* /home bill/school/" places all files starting with algebra string into the school directory. The command has many useful parameters, for example "cp -a" can beused to copy the whole directory trees. Without the -a parameter, copy will not enter subdirectories.

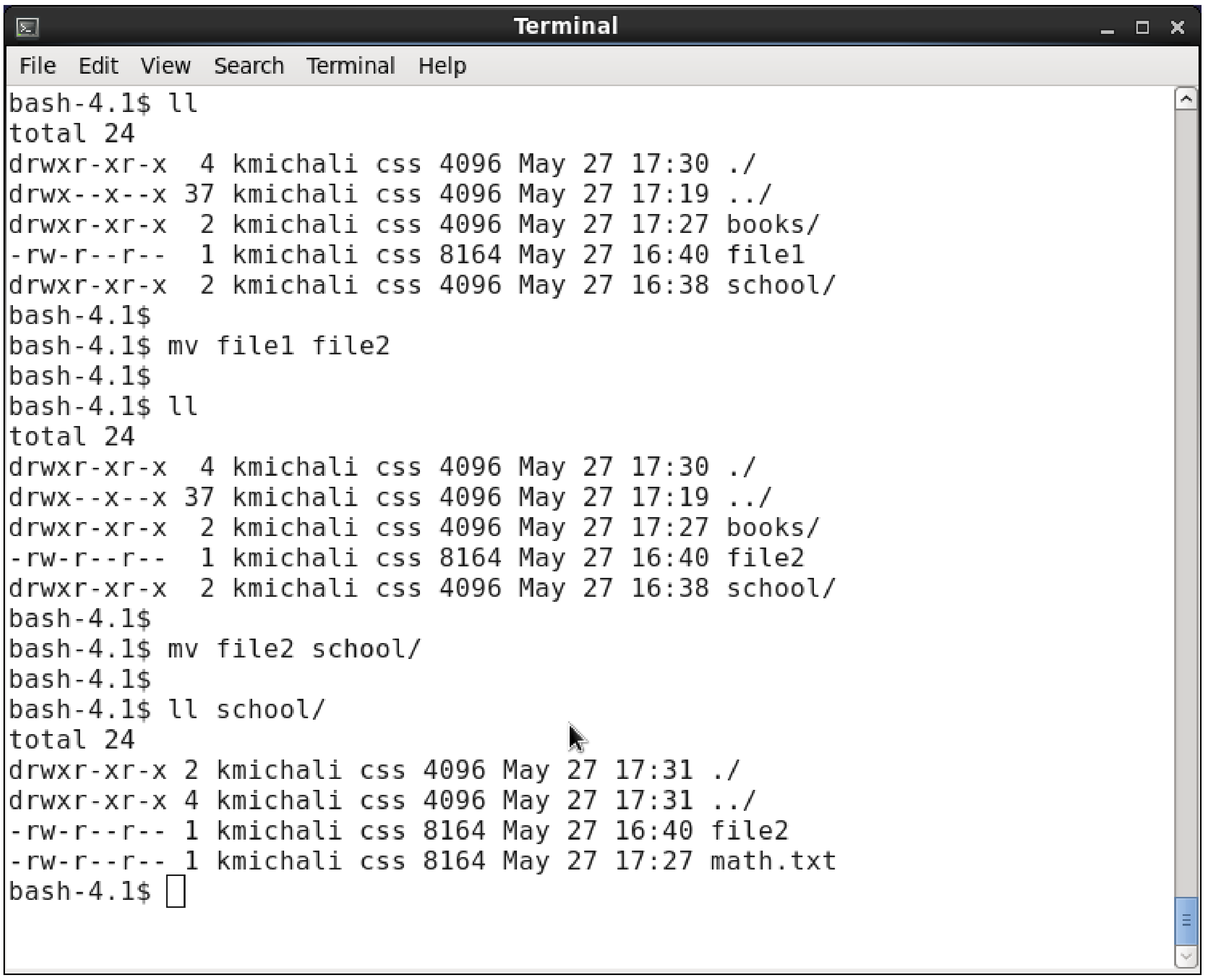
The discussed examples are shown below.



```
bash-4.1$ ll
total 24
drwxr-xr-x  4 kmichali css 4096 May 27 17:26 ./
drwx--x--x 37 kmichali css 4096 May 27 17:19 ../
drwxr-xr-x  2 kmichali css 4096 May 27 17:26 books/
-rw-r--r--  1 kmichali css 8164 May 27 16:40 file1
drwxr-xr-x  2 kmichali css 4096 May 27 16:38 school/
bash-4.1$
bash-4.1$ cp file1 file2
bash-4.1$
bash-4.1$ ll
total 32
drwxr-xr-x  4 kmichali css 4096 May 27 17:27 ./
drwx--x--x 37 kmichali css 4096 May 27 17:19 ../
drwxr-xr-x  2 kmichali css 4096 May 27 17:26 books/
-rw-r--r--  1 kmichali css 8164 May 27 16:40 file1
-rw-r--r--  1 kmichali css 8164 May 27 17:27 file2
drwxr-xr-x  2 kmichali css 4096 May 27 16:38 school/
bash-4.1$
bash-4.1$ cp school/math.txt books/
bash-4.1$
bash-4.1$ ll books/
total 16
drwxr-xr-x 2 kmichali css 4096 May 27 17:27 ./
drwxr-xr-x 4 kmichali css 4096 May 27 17:27 ../
-rw-r--r-- 1 kmichali css 8164 May 27 17:27 math.txt
bash-4.1$
```

Move a file – mv

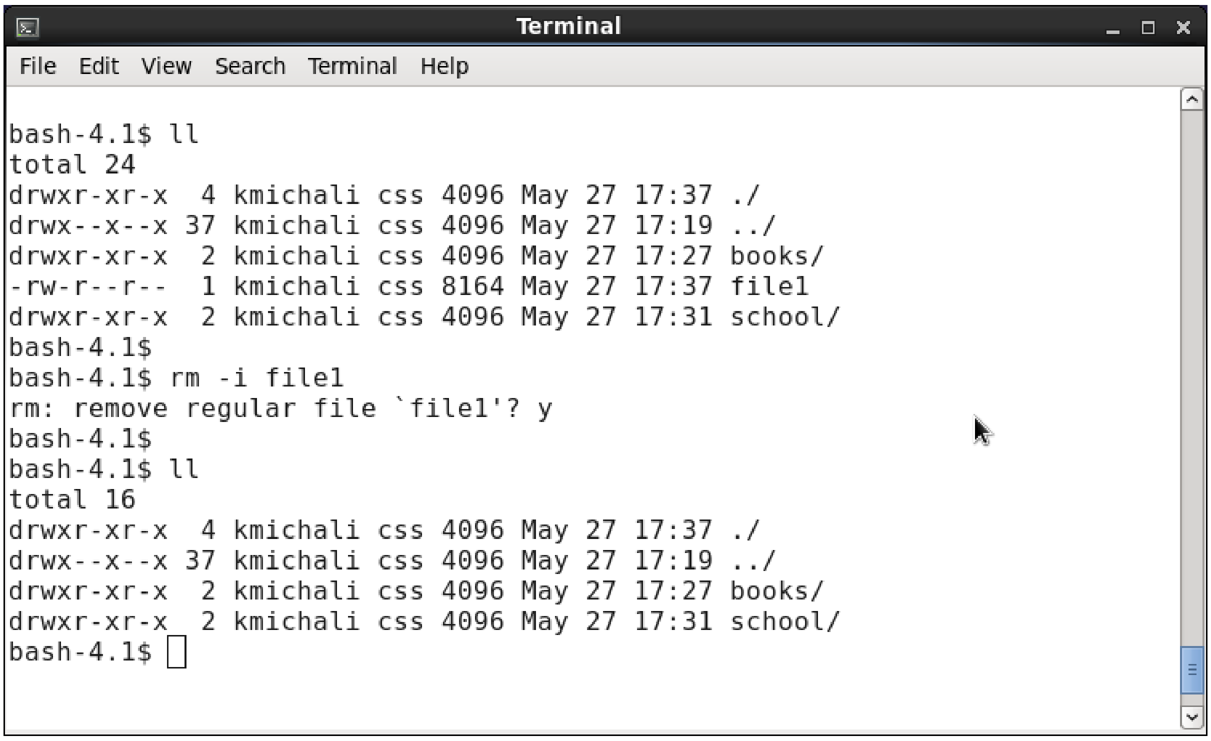
The "mv" (move) command works similarly as rename. The same as the copy command, it requires a target and a destination paths, e.g. "mv file1 file2" renames file1 into file2.   The move command can be also used to relocate a file, "mv /home/bill/file2 /home/bill/school" moves the file2 from "/home/bill" to "/home/bill/school" directory. Move command performed on the same disk is very fast, it involves only a change in file metadata. Move between disk systems involves a copy and delete step.



```
bash-4.1$ ll
total 24
drwxr-xr-x  4 kmichali css 4096 May 27 17:30 ./
drwx--x--x 37 kmichali css 4096 May 27 17:19 ../
drwxr-xr-x  2 kmichali css 4096 May 27 17:27 books/
-rw-r--r--  1 kmichali css 8164 May 27 16:40 file1
drwxr-xr-x  2 kmichali css 4096 May 27 16:38 school/
bash-4.1$
bash-4.1$ mv file1 file2
bash-4.1$
bash-4.1$ ll
total 24
drwxr-xr-x  4 kmichali css 4096 May 27 17:30 ./
drwx--x--x 37 kmichali css 4096 May 27 17:19 ../
drwxr-xr-x  2 kmichali css 4096 May 27 17:27 books/
-rw-r--r--  1 kmichali css 8164 May 27 16:40 file2
drwxr-xr-x  2 kmichali css 4096 May 27 16:38 school/
bash-4.1$
bash-4.1$ mv file2 school/
bash-4.1$
bash-4.1$ ll school/
total 24
drwxr-xr-x 2 kmichali css 4096 May 27 17:31 ./
drwxr-xr-x 4 kmichali css 4096 May 27 17:31 ../
-rw-r--r-- 1 kmichali css 8164 May 27 16:40 file2
-rw-r--r-- 1 kmichali css 8164 May 27 17:27 math.txt
bash-4.1$
```

Delete a file - rm

A file can be deleted using "rm file" (remove). Remove command is irreversible, the file is deleted permanently. There is no trash bin concept when using the command line. It is recommended to use "rm -i" instead of just "rm", this turns on interactive mode, the remove command first asks for a confirmation. Remove command can be also used recursively to delete a whole directory tree, this feature must be used with extreme caution. Rmdir removes an empty directory.



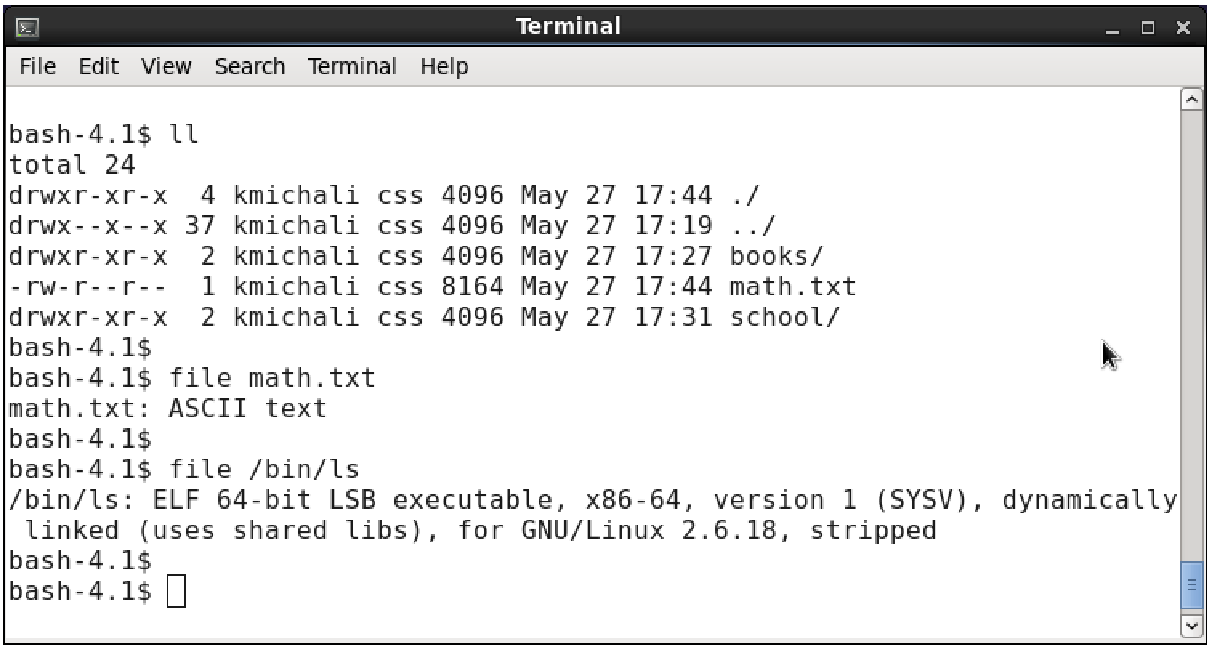
```
bash-4.1$ ll
total 24
drwxr-xr-x  4 kmichali css 4096 May 27 17:37 ./
drwx--x--x 37 kmichali css 4096 May 27 17:19 ../
drwxr-xr-x  2 kmichali css 4096 May 27 17:27 books/
-rw-r--r--  1 kmichali css 8164 May 27 17:37 file1
drwxr-xr-x  2 kmichali css 4096 May 27 17:31 school/
bash-4.1$
bash-4.1$ rm -i file1
rm: remove regular file `file1'? y
bash-4.1$
bash-4.1$ ll
total 16
drwxr-xr-x  4 kmichali css 4096 May 27 17:37 ./
drwx--x--x 37 kmichali css 4096 May 27 17:19 ../
drwxr-xr-x  2 kmichali css 4096 May 27 17:27 books/
drwxr-xr-x  2 kmichali css 4096 May 27 17:31 school/
bash-4.1$ 
```

Text and binary files

Generally speaking, there two types of files when working on the command line.

- text files that are human-readable (e.g. FASTA sequences)
- binary files that contain many more special characters and not easily readable (e.g. a computer program)

The command “file” returns information about the file type. In the example below, "file math.txt" returns "ASCII text" indicating that the file is a text file. If file is applied to an "ls" command “file /bin/ls” the return indicates the binary type.



Terminal

File Edit View Search Terminal Help

```
bash-4.1$ ll
total 24
drwxr-xr-x  4 kmichali css 4096 May 27 17:44 ./
drwx--x--x 37 kmichali css 4096 May 27 17:19 ../
drwxr-xr-x  2 kmichali css 4096 May 27 17:27 books/
-rw-r--r--  1 kmichali css 8164 May 27 17:44 math.txt
drwxr-xr-x  2 kmichali css 4096 May 27 17:31 school/
bash-4.1$
bash-4.1$ file math.txt
math.txt: ASCII text
bash-4.1$
bash-4.1$ file /bin/ls
/bin/ls: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically
 linked (uses shared libs), for GNU/Linux 2.6.18, stripped
bash-4.1$
bash-4.1$ 
```

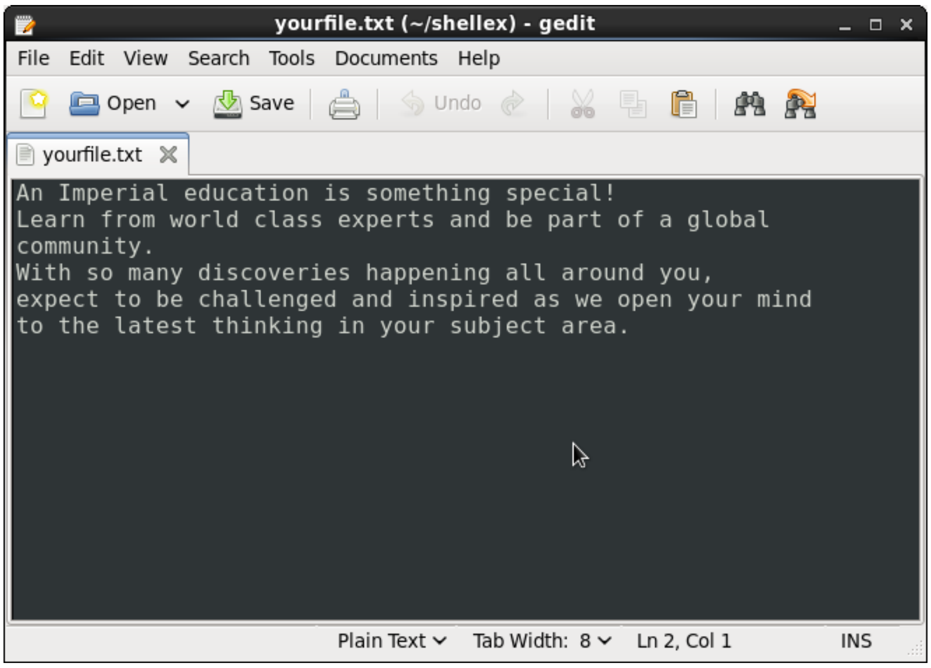
Display content of a file

There are several commands for displaying a content of a text file:

- cat - outputs contents of a file onto the screen
- less - allows scrolling up and down the file using arrow keys and it allows searching (type “q” to quit the less command)
- more - shows one page at a time, press space bar for more pages

Edit content of a file

Command line editors are not graphic based, they have a simple interface and react only to keyboard keys. However, with a good knowledge of commands and short cuts, command line editors can be very effective. This tutorial introduces the nano editor as a good starting point. For more advanced users, there are more complicated editors such as emacs or vi. Emacs has a graphical interface. In the class, we will use a simple graphical editor - gedit; type gedit file_name to open a file for editing.


yourfile.txt (~/shellex) - gedit
File Edit View Search Tools Documents Help
Open Save Undo
yourfile.txt

```
An Imperial education is something special!
Learn from world class experts and be part of a global
community.
With so many discoveries happening all around you,
expect to be challenged and inspired as we open your mind
to the latest thinking in your subject area.
```

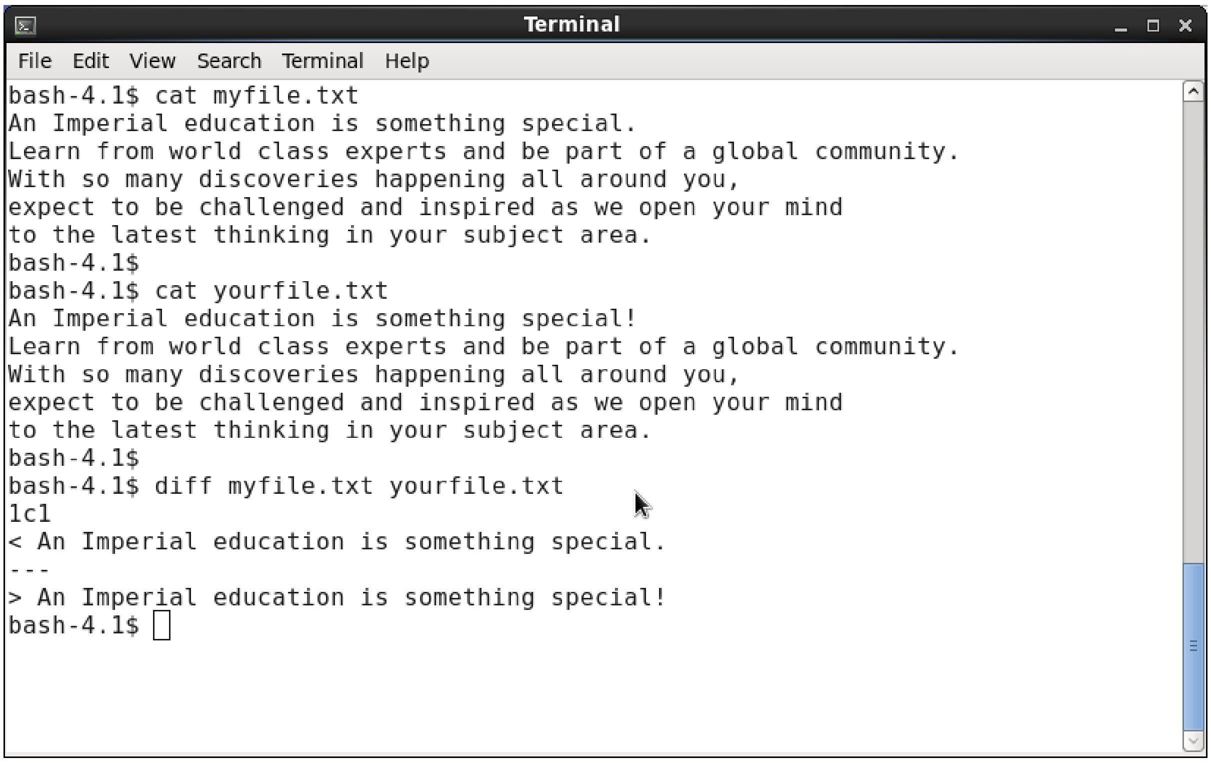
Plain Text
Tab Width: 8
Ln 2, Col 1
INS

Selected Linux commands

Diff

Diff is used for comparing file content. It can be used for text and binary files. In case of text files, it outputs all lines that differ. In case of binary files, it reports only overall result, i.e. the files differ or not. Diff can also be used on directory subtrees, then it reports list of files that differ.

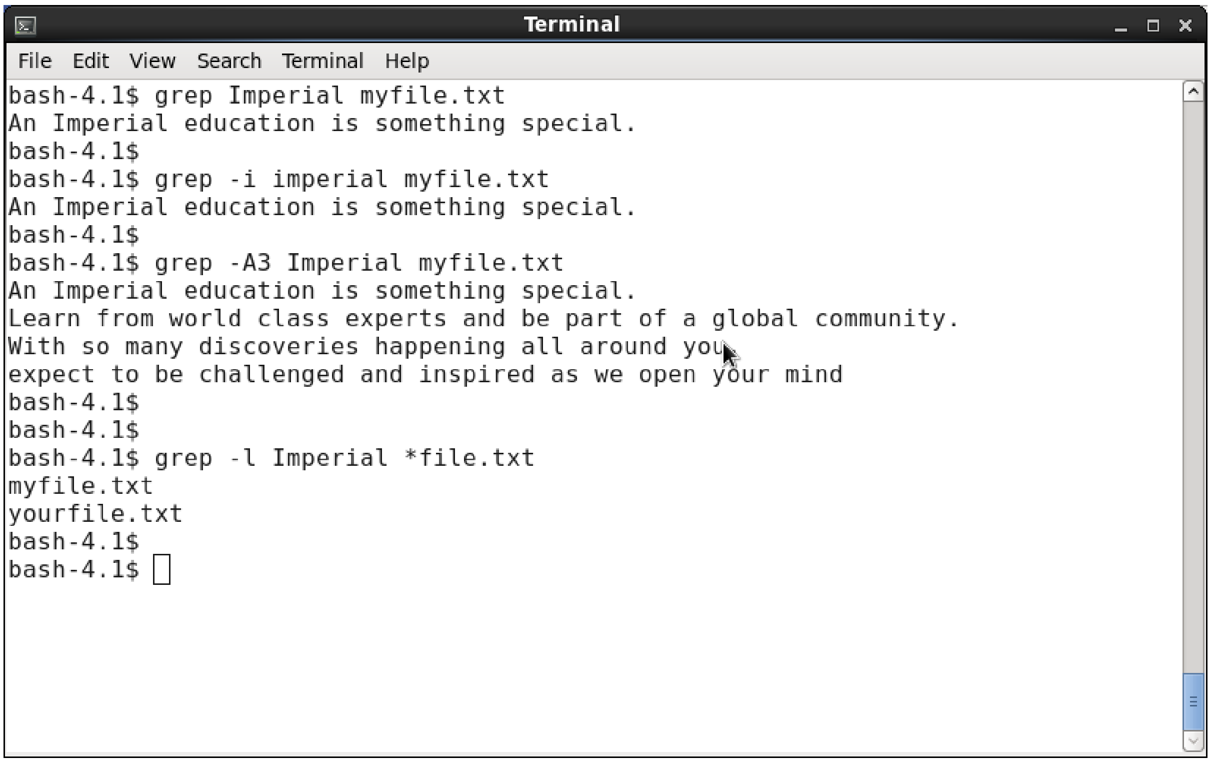
The figure below shows a simple diff between two text files myfile.txt and yourfile.txt. The output shows that there is a difference in the first line.



```
bash-4.1$ cat myfile.txt
An Imperial education is something special.
Learn from world class experts and be part of a global community.
With so many discoveries happening all around you,
expect to be challenged and inspired as we open your mind
to the latest thinking in your subject area.
bash-4.1$
bash-4.1$ cat yourfile.txt
An Imperial education is something special!
Learn from world class experts and be part of a global community.
With so many discoveries happening all around you,
expect to be challenged and inspired as we open your mind
to the latest thinking in your subject area.
bash-4.1$
bash-4.1$ diff myfile.txt yourfile.txt
1c1
< An Imperial education is something special.
---
> An Imperial education is something special!
bash-4.1$
```

Grep

Grep is used to find a pattern in a file. If the pattern is found, grep outputs the whole line containing the pattern. The figure below demonstrates a few useful grep switches. The default behaviour requires exact match. The "-i" switch turns off case sensitivity. The "-l" switch changes the output from the match line to the file name only. Grep can be used with regular expressions using –E flag, for example, grep –E [0-9] matches a string containing only numbers.

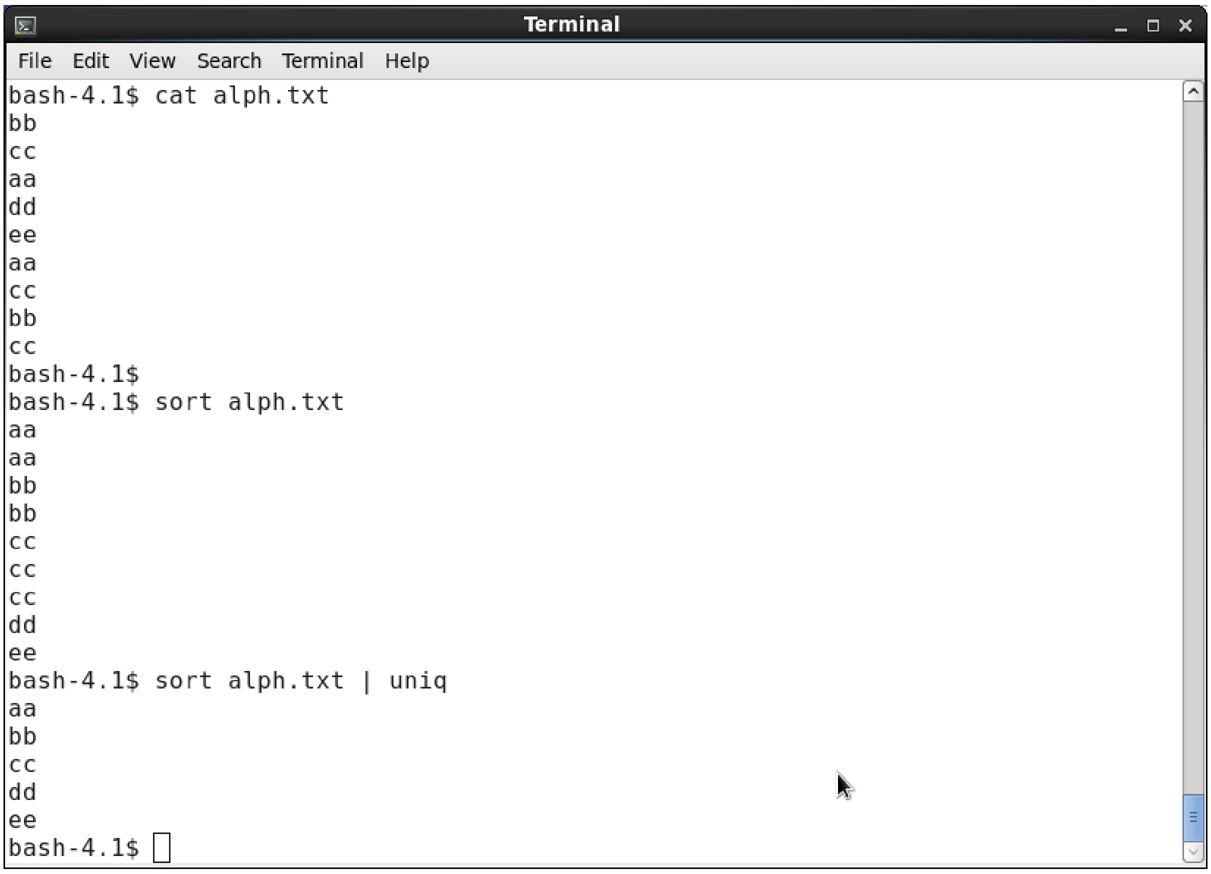


```
bash-4.1$ grep Imperial myfile.txt
An Imperial education is something special.
bash-4.1$
bash-4.1$ grep -i imperial myfile.txt
An Imperial education is something special.
bash-4.1$
bash-4.1$ grep -A3 Imperial myfile.txt
An Imperial education is something special.
Learn from world class experts and be part of a global community.
With so many discoveries happening all around you
expect to be challenged and inspired as we open your mind
bash-4.1$
bash-4.1$
bash-4.1$ grep -l Imperial *file.txt
myfile.txt
yourfile.txt
bash-4.1$
bash-4.1$
```

Sort and uniq

The sort command sorts contents of the file by lines. The default sorting is alphabetical, sort -n turns on numerical sort. The uniq command removes any repeated lines from a file.

The uniq command does not work on unsorted file. The figure below combines the two commands using a "pipe" utility that will be described below.



```
bash-4.1$ cat alph.txt
bb
cc
aa
dd
ee
aa
cc
bb
cc
bash-4.1$
bash-4.1$ sort alph.txt
aa
aa
bb
bb
cc
cc
cc
dd
ee
bash-4.1$ sort alph.txt | uniq
aa
bb
cc
dd
ee
bash-4.1$
```

History

The history prints out all commands previously executed in a shell.

Wc

Count lines (-l), words (-w), characters (-m) in a file (or in an output of a previous command). For example, command ls –l | wc –l will return a number of items in a directory.

Wget

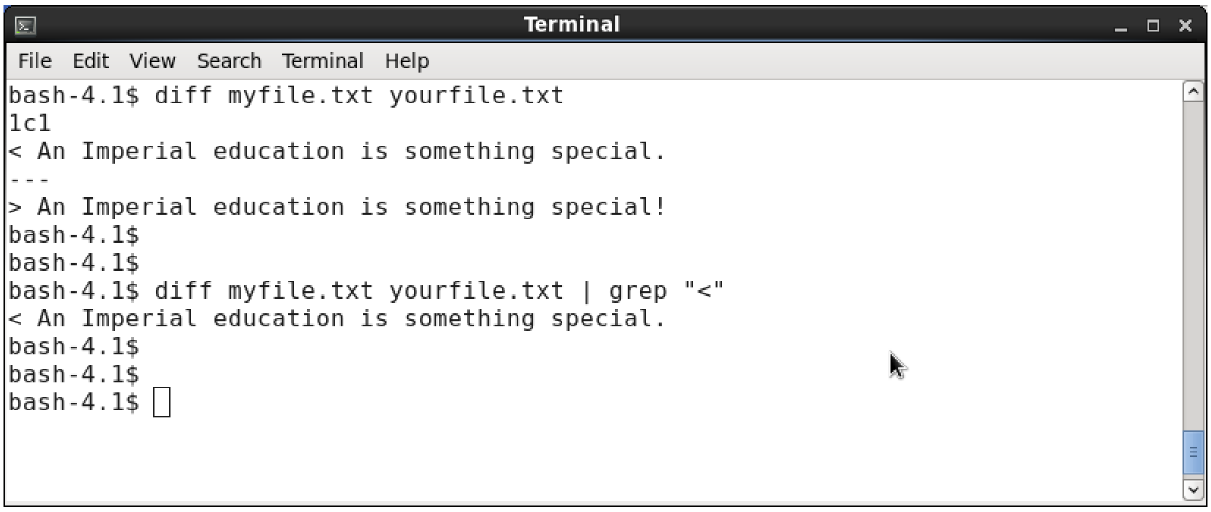
Download file from the web using http, https or ftp protocols.

Syntax example: wget https://bitbucket.org/ImperialHPSC/m3c2015/get /b58b4c99ac27.zip

Pipeline

Pipeline utility takes an output of a command and passes it to a second command as an input. This is accomplished using the “|” pipe sign between commands. The pipeline can be repeated multiple times.

The example below demonstrates use of pipeline between the diff and grep commnands. Grep is used to parse the output of diff ("diff myfile.txt yourfile.txt") and output only the lines that changed in the first file (grep "<").

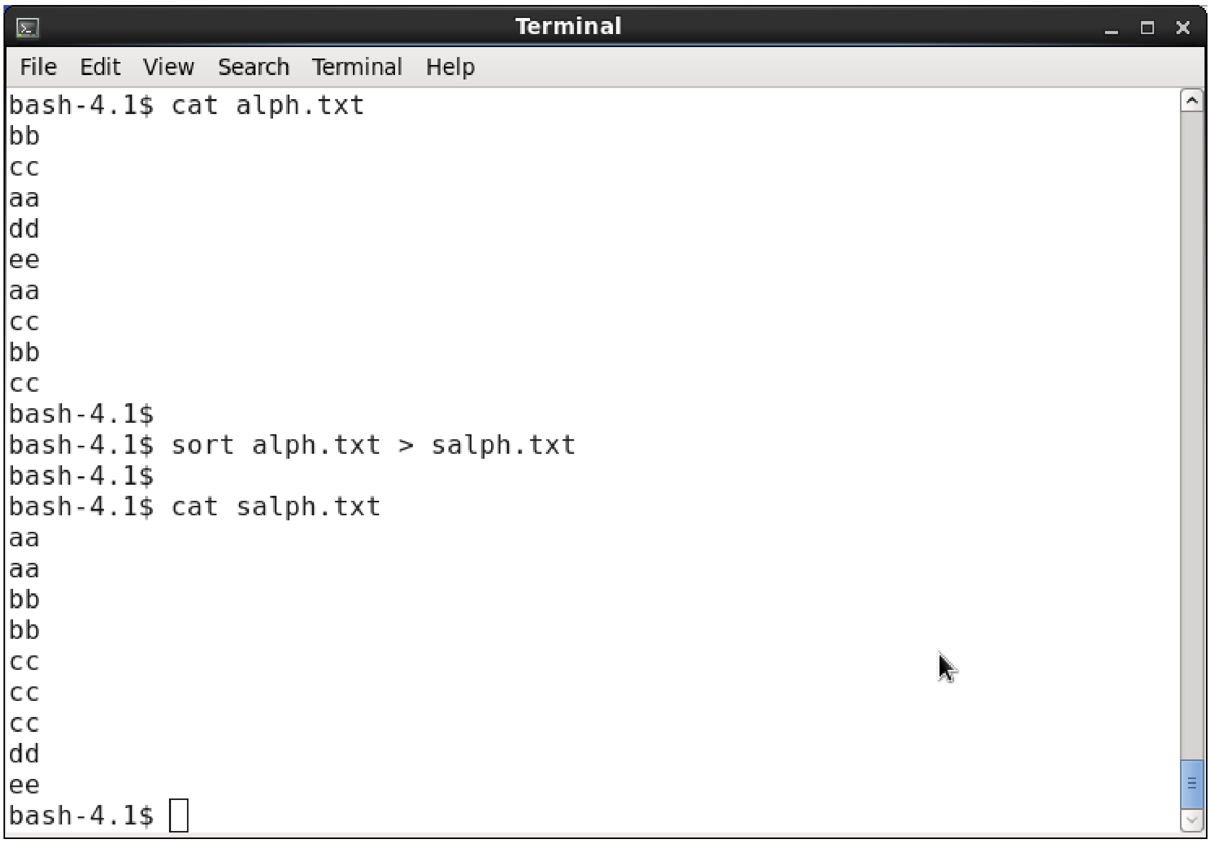


```
bash-4.1$ diff myfile.txt yourfile.txt
1c1
< An Imperial education is something special.
---
> An Imperial education is something special!
bash-4.1$
bash-4.1$
bash-4.1$ diff myfile.txt yourfile.txt | grep "<"
< An Imperial education is something special.
bash-4.1$
bash-4.1$
bash-4.1$ 
```

Redirecting input/output

If your software or a command does not provide an option for storing results in a file and prints to the screen, use greater than sign “>” to redirect the output to a file. The statement “command > file_name” stores the command output in a file.

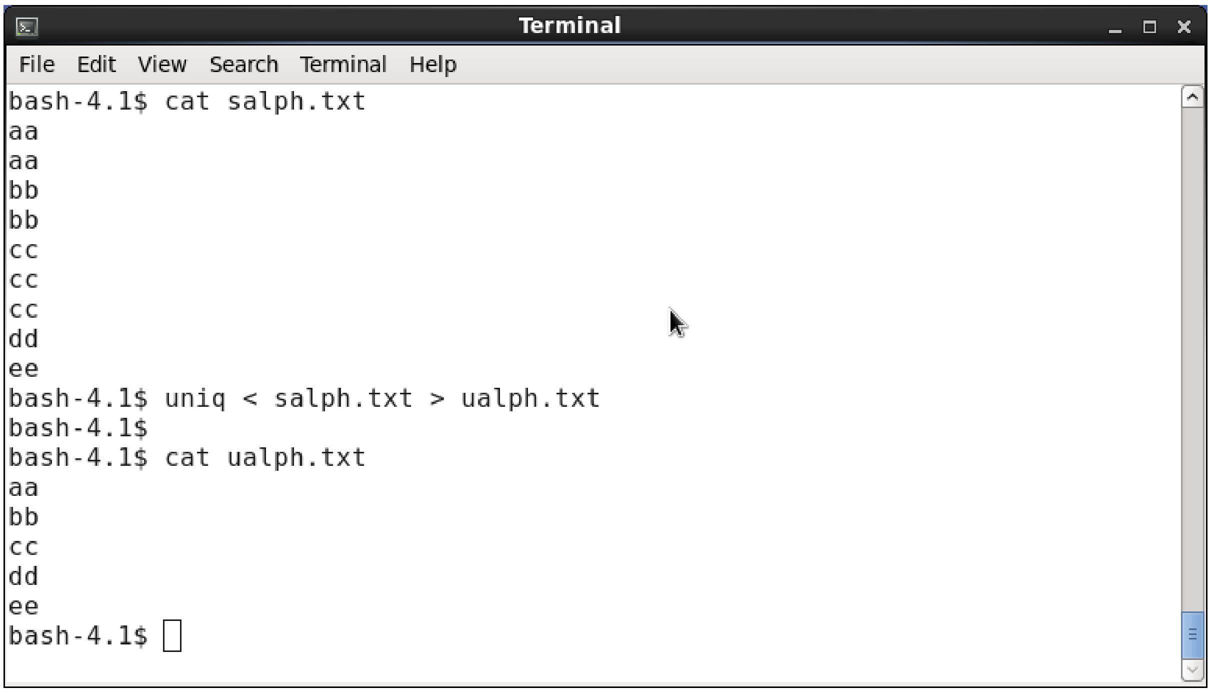
The example below performs sort on the file alph.txt. The output is redirected to the file salph.txt using the ">" sign.



```
bash-4.1$ cat alph.txt
bb
cc
aa
dd
ee
aa
cc
bb
cc
bash-4.1$
bash-4.1$ sort alph.txt > salph.txt
bash-4.1$
bash-4.1$ cat salph.txt
aa
aa
bb
bb
cc
cc
cc
dd
ee
bash-4.1$
```

One can also reverse this and "feed" an input to a command with less than “<” sign. The construct “command < infile” executes the command on the infile. The construct “command < infile > outfile” combines both methods. The command is executed on infile and the output captured to outfile.

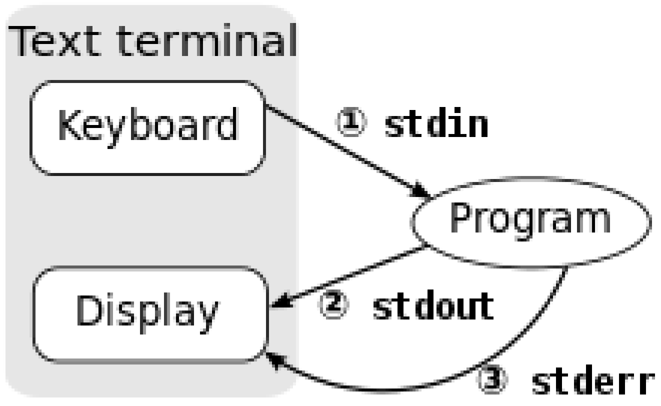
The construct "uniq < salpth.txt > ualph.txt" takes the file "salph.txt" as an input to the uniq command and redirects the output to the file named "ualpth.txt".



```
bash-4.1$ cat salph.txt
aa
aa
bb
bb
cc
cc
cc
dd
ee
bash-4.1$ uniq < salph.txt > ualph.txt
bash-4.1$
bash-4.1$ cat ualph.txt
aa
bb
cc
dd
ee
bash-4.1$
```

Standard streams

Standard input and output are so-called standard streams. These are predefined channels of communication between a program and its environment. The figure below illustrates the concept. Stdandard input (stdin) comes from the keyboard while stdout is streamed to the diplay by default. The third standard stream is standard error - stderr.



Some programs stream error messages to stderr to separate errors from the data output. The construct "command > log 2 > &1". What does this mean? On the command line, one can use "&0" to refer to stdin, "&1" to stdout and "&2" to stderr. Using this notation, the above expression redirects the program output to the "log" file. The "2 > &1" at the end of the expression redirects the standard error (2) to standard output (1). This way the log file contains the program output as well as possible errors.

Environment variables

Apart from user-defined (or local) variables, there are environment variables that are always set in the current shell. Type "printenv" to see all environment variables. To mention a few, the home directory path is set in the HOME variable, the current working directory is set in PWD, the user name in USER and the shell type in SHELL.

PATH

The PATH variable contains a list of directories that are searched for executable files every time a user issues a command. If a command is located with the list of directories, it can be called simply by its name. If a command is not located, the full path has to be used to evoke it.

Commands located in current directory often confuse new users. The thinking is that if the command is located in the current directory and has the right permissions, one should be able to simply type the command to execute it. If this works or not depends on the way the PATH variable is set.

In the example below, the PATH contains several directories. However, none of the directories is the current directory "./". When the command is called with just a name "first.sh" shell produces an error "command not found" because the current directory is never searched. For the command to be found, it would have to be called with the path to the current directory "./first.sh".

```
bash-4.1$ echo $PATH
/usr/lib64/qt-
3.3/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/sbin
bash-4.1$
bash-4.1$ first.sh
bash: first.sh: command not found
bash-4.1$ ./first.sh
This is my first script!
bash-4.1$
```

Personalise your shell .bashrc

Alias

The alias command is used to set up an alias to a standard unix command. For example, many systems have alias for ls –l already set as ll. If your machine does not recognise “ll”, add alias ll=”ls -la” into your .bashrc.

PATH

It might be useful to add extra paths to PATH variable. You can modify your path:

```
export PATH=$PATH:/another/directory
```

Example .bashrc file:

```
bash-4.1$ more ~/.bashrc
#--------------------
# Personal Aliases
#--------------------

alias ll='ls -lahF'
alias lt='ls -laht'
alias lS='ls -lahS'

alias du='du -kh'
alias df='df -kTh'


#--------------------
# Prompt set up
#--------------------

PS1="\u@\h \w> "


#--------------------
# Environment set up
#--------------------

export LD_LIBRARY_PATH=/usr/lib64/openmpi/lib:$LD_LIBRARY_PATH
```

Other resources

Unix tutorial: http://www.ee.surrey.ac.uk/Teaching/Unix/index.html

Advanced materials: http://tldp.org/guides.html

Murder mystery home: https://github.com/veltman/clmystery