The command line interface

Kovács Kristóf, Magyar András, Simon András

BME TTK Matematika Intézet

December 6, 2022

To goal is to be able to use the computer infrastructure of the Institute (which is 90% Linux).

- `Username:` XXX
  `Passwd:` XXX
- `http://wiki.math.bme.hu/view/Informatics1-2017/Practice1`
  Lots of practical advice. For example how to log in from home, how to use the wifi in the Institute.

Az excerpt from a Linux filesystem:

```
/usr
    /usr/bin
    /usr/games
    /usr/include
        /usr/include/arpa
        /usr/include/asm
        /usr/include/asm-generic
        /usr/include/atk-1.0
            /usr/include/atk-1.0/atk
        /usr/include/at-spi-2.0
            /usr/include/at-spi-2.0/atspi
        /usr/include/at-spi2-atk
            /usr/include/at-spi2-atk/2.0
        /usr/include/bits
            /usr/include/bits/platform
            /usr/include/bits/types
        /usr/include/blkid
        /usr/include/brotli
         ...
```

pwd print the name of the working directory (i.e. the one you're in); we can always refer to this directory as `.`

ls list the content of the current working directory. For example:

```
$ ls
```

or

```
$ ls /dev
```

or

```
$ ls -hl /usr/bin
```

cd change directory. Without parameters it takes us to our home directory. For example:

```
$ cd /mnt
```

or

```
$ cd
```

or

```
$ cd ..
```

pushd/popd	`pushd somedir` is like `cd somedir`, except that after issuing this, `popd` takes us back to where we were; these can be nested
mkdir	make directory. For example: `$ mkdir newdir`
cp	copy file(s). For example: `$ cp what.txt towhere.txt` or `$ cp what.txt towhere/` (where `towhere` is a directory, and can also be `.` or `..`). `cp -r` copies recursively — this is how one can copy a complete directory structure

mv move (rename) file. For example:

```
$ mv what.txt towhere.txt
```

rm remove (erase) file. For example:

```
$ rm what.txt.
```

The following is dangerous:

```
$ rm -r somedir
```

because it removes recursively the directory `somedir`.

quota Prints how much is left of the space assigned to us. It's important to keep our usage low, otherwise we won't even get our mail. What we can do in this case is log in a nongraphical terminal (Ctrl-Alt-F5) and delete what we we don't need anymore.

df, du disc free space (how much free space there is on the mounted filesystems), disc usage (how much disc space is consumed by the files in a directory, including all its subdirectories). With the option `-h` their output becomes human readable. For example `$ df -h`.

file managers `mc`, emacs dired mode, . . .

tab completion Pressing TAB completes the names of files, names of commands (and sometimes even names of parameters of commands).

history we can us the up/down cursors to move among older commands (the command history); we can even search incrementally backward among the commands with `Ctrl-r`.

job control Usually `Ctrl-c` can be used to shut down (kill) a program started from the command line; `Ctrl-z` suspends it, and the suspended program can be put in the background with `bg` (meaning the it continues to run, but we get back our ptompt), `fg` puts it back in the foreground. One can start a program in the background by putting the `&` character after the name (and possibly the arguments) of the program. This makes sense with longrunning, non-interactive programs, but these days it's often simpler to open a new terminal window.

cat	`cat` copies the content of its argument (the name of a file) to the terminal. So one can use it to inspect short text files, but this is not its most important use. Using redirection one can conCATenate files with it (see later!).
less	reading long textfile (or outputs of programs); for example `$ less what.txt`. One can navigate it with `Up/Down` and `PgUp/PgDn`.
tail	inspect the end of a text file; with the option `-f` (“follow”) one can watch what gets written at the end of a file. This is useful if a program sometimes writes a new line in a file and we want to know what’s going on.

For example:

```
$ for i in {1..10}; do echo $i » /tmp/logfile; sleep 10; done &
$ tail -f /tmp/logfile
```

wc statistics on text files: it counts the number of bytes (or, with `wc -m`, the number of characters, which is more useful), words and lines; all three by default, but for example

```
$ wc -l text.txt
```

counts only the number of lines

grep filters the lines of a text file according to a pattern; for example,

```
$ grep cake text.txt
```

returns the lines in `text.txt` containing the word "cake". (If we want to know how many such line are there, we can do

```
$ grep cake text.txt | wc -l
```

We'll see shortly why this works.) The first argument of `grep` can be a *regular expression*.

- `>` redirects the output of a command into a file. For example:
  `$ ls -l > hereswhatyouhave`
  puts the content of the working directory in the file `hereswhatyouhave` (replacing whatever was there).
  Another example: after
  `$ cat file1 file2 > file12`
  file12 will be the concatenation of file1 and file2.
- `>>` like >, but appends instead of overwriting
- `|` `command1 | command2` the output of the first command will be the input of the second. This is how `$ grep cake text.txt | wc -l` above works. The output of `grep` is the sequence of lines containg "cake", and `wc -l` counts these. Another example:
  `$ ls -l | less`

Another example from here:
https://datascienceatthecommandline.com/2e/index.html

```
$ curl -s "https://www.gutenberg.org/files/11/11-0.txt" |
  grep " CHAPTER"
 CHAPTER I.     Down the Rabbit-Hole
 CHAPTER II.    The Pool of Tears
 CHAPTER III.   A Caucus-Race and a Long Tale
 CHAPTER IV.    The Rabbit Sends in a Little Bill
 CHAPTER V.     Advice from a Caterpillar
 CHAPTER VI.    Pig and Pepper
 CHAPTER VII.   A Mad Tea-Party
 CHAPTER VIII.  The Queen’s Croquet-Ground
 CHAPTER IX.    The Mock Turtle’s Story
 CHAPTER X.     The Lobster Quadrille
 CHAPTER XI.    Who Stole the Tarts?
 CHAPTER XII.   Alice’s Evidence
```