Tekstmanipulatie, week 9

General Introduction

Introduction to the course. Please, read this page and this page, too.

What is a computer in fact?

- Computing (almost never done in humanities) vs. other tasks: building and manipulating databases, texts, etc. This course will deal with manipulating texts.

  Examples: searches, statistics (number of words, sentences, N-grams, etc.), concordances, systematic transformation of texts,... Machine Translation?
- Is a toaster a computer? And a calculator or an abacus?

Main characteristics of a computer:

- mechanical vs. electronic
- analogue vs. digital
- multi-level organization (hardware vs. software)

Software: feeding the computer with a software is as easy as feeding the computer with the data, and yet it determines the task performed by the computer. Earlier, you had had to rebuild the whole machine, when you wanted to solve a new problem.

John v. Neumann - Herman Goldstine (1948): binary system + controlled by a software program

Several levels between the machine and the user:

- machine code, assembly, assembler: (almost) direct connection to the hardware (not user friendly)
- **operating system**: the level on which the user, the hardware (CPU, the

peripherals: monitor, CD-ROM, floppy disc, keyboard,...) and the software communicate with each other. *An operating system is a program (or a set of programs) that controls the way the computer works, and it runs other programs.* (Examples: DOS, MS Windows 3.1 / 95 / 98/ 2000, OS2, different versions of Unix and Linux, etc.)
- user friendly applications

Why using PC's at post offices or for typing simple letters? The advantages of this kind of modularity are:

- Easier to program
- Portability between different types of computers
- Higher degree of flexibility (e.g. including new options)

... therefore wider market to sell them, so they are cheaper.

Unix

Unix as an operating system. And also a culture, a way of thinking. Starting at AT&T, in 1969... standards and plenty of variations (Linux).

(Miles Osborne's Unix slides .)

Elements of the Unix philosophy:

- Multi-users: one host - many terminals
  - Old type terminals connected to the host
  - Connection to the host via the internet (cf. telnet) or via the phone line.
  - Nowadays: XWindows and virtual terminals (CTRL+ALT+F1-F7: 6 "black-and-white", 1 graphical).
- Multiprocessors, timesharing
- **Shell** (e.g. bash): the program that is launched when you log in or when you open a new terminal, and that looks after the user.
- Standard functions, although different buttons on the keyboard (^h = delete, ^l=rewrite the screen, ^c=stop the command, ^d=end-of-file,...) (**remark**: ^ stands for pushing simultaneously the CTRL-button).
- Standard processes with commands (possibility to redefine them)
- Use the drive, rather than the memory (remember: very low memories in

1969...)
- Use of regular expressions (actually wild cards are also regular expressions)

Unix shells

A shell is really the envelope around the computer: it receives the commands and executes them. How does it work?

- Printing the "prompt" (which may include the machine's name, the actual working directory, the time, etc.) and waiting for a command (or a group of commands) from the input, e.g. from the keyboard (standard input). (In fact, from a file, such as a Shell script or the standard input, which is also a file, ending with ^d... "everything is a file!", now we understand while is ^d = eof = exit!)
- Preprocessing it (e.g. resolving wild cards)
- Executing it (line-by-line: not all errors become obvious already at the beginning!)

Different types of shells [different kinds of prompts]: e.g. Korn shell (ksh) [%], C-shell (Csh) [$], Bourne shell [$], Bourne Again shell (bash) [>]

We most often use the Bash-shell. In order to save typing a lot:

- cursor up: previous commands
- TAB: fills in the file names, if there is only one possibility
- TAB TAB: if there are more possibilities, you can get a list of them by typing TAB a second time

The User

Logging in, login name, password, changing password ('passwd'), logging out ('logout', 'exit', ^d)
A user is given:

- a user name (login name)
- a password
- a home directory

When you log in (login name + password), a shell is launched. This shell looks after you, waits for your commands, and executes them. Finally, you can exit the shell ('logout', 'exit', ^d), but then who takes care of you? If nobody, then you are logged out.

Root privileges (only the system administrator) vs. 'regular users'.

The UNIX file system

What is a file (in Unix "everything is a file")?

- A sequence of bytes (numbers between 0 and 255).
- This sequence of bytes can have different interpretations, based on the application (program) using the file.
- An executable program: a file understood as a sequence of instructions to the computer.
- A plain text file: a file whose bytes are interpreted as characters.
- In Unix, even directories are special files.

Finding the needed file out of thousands ones is usually hard. Therefore, operational systems use a hierarchical tree with a root, branches (= directories) and leaves (= files).

In Unix, the **root directory** is referred to as / . For instance, to list the content of the root directory, type `ls /`.

The symbol / refers also to a subdirectory. Thus, if `blue` is a file in the directory `colours`, then we may refer to it as `colours/blue`. If the directory `bin` is a subdirectory of the root (/), and if within `bin` there is a file named `bash` (it is in fact the program that usually runs as the shell), then we can refer to this file as `/bin/bash`.

To refer to your home directory, use ~. When the system administrator created your account, he also created your home directory (usually within the directory /home or `/users`, therefore your home directory is most probably /home/your_user_name or

/users/your_user_name). When you log in under some user name, then the symbol ~ refers to the home directory of that user.

It is important to remember that the home directory is a crucial border in the file tree between the system and the user: bellow it, you are free to do anything (create sub-directories, files, remove them, etc.). However, you cannot do anything outside of your own home directory. Only the superuser (the system administrator), who has "root privileges", is able to make changes in the whole file system (including the root directory, hence the expression "root privileges").

In any moment, there an *actual working directory*. This is the point in the file system tree where you are now. You can refer to it as . . Relative to some point in the tree, the symbol . . refers to the parent directory (looking at one level upper in the tree).

Paths: a path is the way you identify the place of a given file in the file structure (don't forget that files at different points of the file structure may have the same name)

- Absolute path: the complete path from the root (/) on. This path is always the same for a given file, and does not depend on under which name you have logged in or what is your current working directory. Example: /home/s12345/tekstmanipul/colours/blue.
- A path relative to the home directory (~/) (depends on the login name used when logging in). Example: ~/tekstmanipul/colours/blue.
- Relative path: you start from the actual working directory (. / or simply nothing). An example (it may be useful to draw it to yourself on a sheet of paper):

If you are in the directory tekstmanipul within your home directory, and you have logged in under the user name s12345, then the file /home/s12345/tekstmanipul/colours/blue can be reached as ./colours/blue or simply as colours/blue. If you are, however, in the directory ~/algorithm, then you have first to walk one level up, before moving down the tree: ../tekstmanipul/colours/blue. Furthermore, if your working directory is ~/algorithm/week1, then you have to work up two levels to reach the required file blue: ../../tekstmanipul/colours/blue.

Summary:

/ : the root directory
__/__: subdirectories (cf. \ in DOS!)
~: home directory
. : the actual working directory
.. : parent directory
../.. : grand-parent, etc.

/abc : a file in the root directory
~/abc : a file in my home directory
./abc : a file in the actual working directory
../abc : a file in the parent directory of the actual working directory.

Important commands relating to the file structure:

'pwd' : **p**rint **w**orking **d**irectory (to file)
'ls' : **l**i**s**t
mkdir : **m**a**k**e **dir**ectory
cd : **c**hange (working) **d**irectory
cat: catenate (concatenate) (lat. catena = 'chain') : use now this to create the simplest files.

A few important remarks about file names:

- *UNIX is case sensitive, therefore 'tamas', 'Tamas', 'TAMAS' or 'tamAs' would be four different file names!*
- *No extension (like .exe, .bat, .doc or .rtf) exists in UNIX; therefore the period ('.') is just considered to be one character of the file name, therefore 'my.first.file.name' would be a legitimate file name in UNIX. It is up to you to find out a file naming system that you find useful. Some newer programs do use extensions (the character string after the* last *period), but this is independent from the UNIX operational system.*
- *No constraints exist in UNIX for naming files. But you should consider not having too short, neither too long file names, but names that are informative enough and easy to handle.*
- *Furthermore, you do better not use the following characters in your file names: space, coma, /, (,), ', ", +, *, ?, <, >, $, \, and avoid using '-' as the first character, because these characters will have special meanings. (You will have to use escapes to avoid the shell interpretting these characters as having special meanings.)*

Different types of files:
- data files: not executable, but programs can read and

/ or write (re-write) them.
- program files: executable files (see also permissions)
- directories (yes, even directories are a special type of file)
- links , and other special types of links (e.g. the peripherals, like the floppy disc, the keyboard or the screen, are also files).

An important principle in Unix: EVERYTHING IS A FILE!

Directories are files. (This principle will be very important when we will learn about pipe lines.)
The "screen" is also a file (e.g. /dev/tty1 or /dev/pts/1): you can write something to a terminal (or virtual terminal) by writing into that file. (We will come back to this when we will learn about pipe lines.)
Drives are also files (directories) within the same hierarchy (e.g. /media/floppy/) (unlike DOS, like Windows).

The command mount virtually connects an external device (for instance a floppy disk) to the file system, and the command umount removes is. Thus, if you want to use a floppy disk, first you have to mount if with `mount /media/floppy` (the exact path to be used depends on the system). Then, the files on your floppy are under `/media/floppy` (or something slightly different, depending on the system). For instance, if your floppy contains a directory `assignments`, and within it, there is a file `assignment1`, then this file is `/media/floppy/assignments/assignment1` for Unix. At the end, you have to unmount the floppy by using the command `umount /media/floppy` (or something slightly different, depending on your system's configuration). Unmounting the floppy before removing it from the drive and / or before logging out is necessary for three reasons, at least. First, maybe this is only when the new information is put physically on the disk (before that, the system kept track of what should have been on the disk). Second, if you do not unmount your floppy, you are not able to use another one. Last, but far not least, if log out without unmounting your floppy, other people cannot unmount yours, and they will not be able to use their own floppy disks.

Unix commands

Commands are either functionalities of the running shell, or program

files that you can simply launch from the shell.

The general syntax of them is:

command [-options] [arguments]

Remark: [...] always means that this part is optional. What means e.g. abc [de [fg]] ?

Getting help / information about a given command:

man <command's_name>
<command's_name> - -help | less

The second option uses the *online manual*, a very useful functionality of Unix. If you do not know the command's name that you are looking for, you can use `man -k` *keyword* to look for a keyword in the online manual (probably, you will use `man -k` *keyword* `| less`).

Furthermore, it is important to know that the online manual has many chapters (usually referred to as numbers in brackets, e.g. in the "see also" section). If you type simply man *command*, you will receive the first description appearing in the online manual. So, for instance, if you type `man time`, Unix shows you an entry from section 1. However, if you type `man 2 time`, you are returned a very different description, from section 2. (The latter is actually a C function, because the online manual also includes information about the C language, a programing language closely related to Unix)

Remark: the command man uses the program `less` in order to visualize the online manual. To scroll down in either `man' or `less', just press <SPACE> or the cursor buttons. To leave these programs, press Q.

Basic commands

Getting help / information about a given command:

man <command's_name>
<command's_name> - -help | more

We will use the command 'cat' to create and to read files, although we will discuss it only next week:

cat > file_name puts the text we are typing into the file, and we finish the file by typing ^d (where ^ refers to CTRL)
cat file_name gives us the content of that file.

passwd : change your password

logout : logging out (on Linux graphical terminals: use the logging out button instead)

exit : quit the shell (practically speaking, it means very often logging out)

cd : change (working) directory
Going home with 'cd' or 'cd ~'

pwd : print working directory

who : who are logged in?
'whoami' or 'who am i'

date : gives the actual date and time

cal [month] year : gives the calendar of that given month / year. Try 'cal 9 1752', and remember what you know about the Julian and the Gregorian calendar!

ls : list

ls lists the actual working directory
ls -l gets long list
ls <file_names> lists the given files or directories
ls -R lists recursively the subdirectories
etc.

Always list the directory you have just done something!
Check yourself after every step!

mkdir : make directory

mkdir <dir_name> [<dir_name>...]

rm, rmdir : remove files and directories (watch out!)

rm -r : removes the subdirectories recursively
rm -i : asks for affirmation

cp : copy
mv : move (copy + delete the original one)

cp / mv [options] <origin> <destination>
The destination can be both a file name or a directory. If the two arguments of cp or mv are regular files, then the file to which the first argument points is copied or moved to the file to which the second argument points. Previously existing files are overwritten. However, if the last argument is an already existing directory, then the source file is copied or moved *into* that directory, and its name is kept. Furthermore, if the last argument is a directory, you can have more than two arguments: all the files which are referred to by the first arguments (all arguments except the last one) are moved into that directory. If the last argument is not a directory, the action does not make sense and you receive an error message: which of the first files should be copied or moved to the last one?

How to rename a file? Renaming a file is nothing but moving file x to file y: you delete file x, and you create a new file, with the name y and with the same content. If you do it within the same directory, then it looks as if you have renamed the original file. Consequently, you the command `mv' to rename a file.

Examples (try to understand them! observe how relative and absolute paths are used!):

- mv old_name new_name : the file `old_name' is renamed to `new_name'.
- `cp ../John Jack` : a new file is created in the actual working

directory, whose name is `Jack', and whose content is identical to the content of the file `John' which is located in the parent directory.

- `cp /bin/cp /bin/ls /bin/mv ~/my_commands` : three files from the directory `bin' (which is a subdirectory of the root directory), namely the files containing the commands `cp', `ls' and `mv', are copied to the directory `my_commands' that is a subdirectory of my home directory. If, however, I have not created the directory my_directory within my home directory before hand (using `mkdir my_directory'), I receive an error message ("copying multiple files, but last argument `/users/username/my_commands' is not a directory").
- `mv Mary ..` : the file `Mary', which is located in the current working directory, is moved one level up in the tree structure (hence, the `..` characters).
- `mv ../../Willy .` : the file `Willy' has been located in the "grand mother directory", but now it is moved to the actual working directory (where I am now), which I refer to by the `.` character.

***A very very important rule** : if a file has been removed, there is no possibility forever to recover it, unlike in the case of DOS!!! (Remember, Unix is designed for multiple users working parallel in the same file structure.)*

Bíró Tamás:
e-mail
English web site
Magyar honlap

Last modified: Thu Jul 3 11:39:17 METDST 2003