

Computer Systems

3. Programming



Review

- Computers, signals, information, their representation
- Representation of numbers
- Fix-point, floating-point representation
- Codewriting
- ASCII, UTF-8 code-tables
- Architecture, the role of operating systems
- Clients-Server differences
- Graphical and character connections
- File system, base commands

What follows next ...

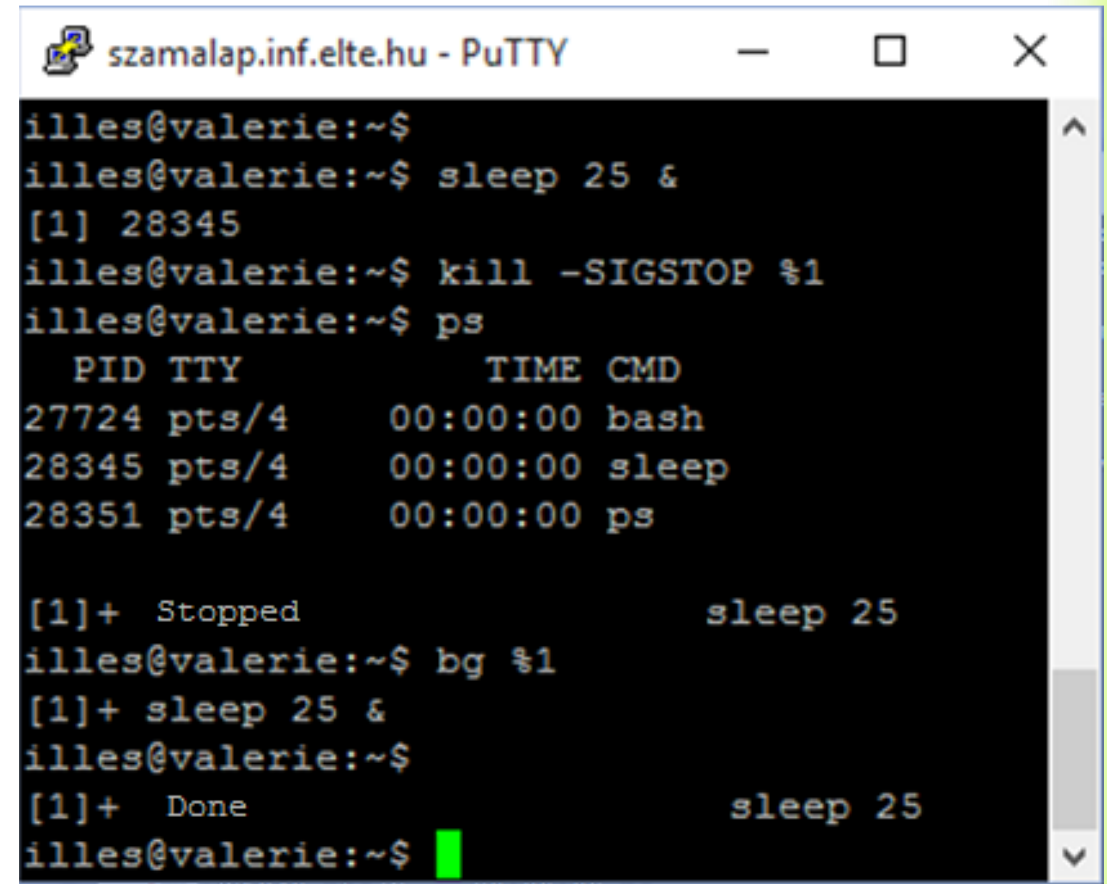
- Further commands
- Processes in the foreground and in the background
- I/O redirection
- Filters
- Let's start programming!
- Regular expressions!

Further commans?

- Commands in the foreground: normal command execution
- Command execution in the background: & character at the end of the command
 - & namings: ampersand, and, andsymbol, atsymbol
- Sleep 15 # waits 15 seconds, during this period there is no possibility to run new commands
 - CTRL+Z sends a stop signal to the process
- sleep 15 & # the sleeping process is running in the background, during this we can give new commands
 - As a result we get: [1] 28321, the first number is the „jobnumber“, the second one is the PID (processid)
- jobs—lists out our running commands(our „jobs“)
- ps—process status command, we can see the sleep running in the background too

Processes in the background

- To replace a background process into the foreground use: `fg %1`
- After stopping a foreground process(with `ctrl-z`) we can continue it in the background too: `bg %1`
- To send a signal: `kill–signal process`
 - The process is identified by it's job number(must use% symbol) or by its pid (process identifier) number.



```
illes@valerie:~$
illes@valerie:~$ sleep 25 &
[1] 28345
illes@valerie:~$ kill -SIGSTOP %1
illes@valerie:~$ ps
  PID TTY          TIME CMD
 27724 pts/4        00:00:00 bash
 28345 pts/4        00:00:00 sleep
 28351 pts/4        00:00:00 ps

[1]+  Stopped                  sleep 25
illes@valerie:~$ bg %1
[1]+  sleep 25 &
illes@valerie:~$
[1]+  Done                      sleep 25
illes@valerie:~$
```

Much more about processes

- top command: you can see the data about running processes, the state of the global system!
- Everybody is equal?
- Unix, Linux system priorities between -20, 19 (40 levels)
- The smaller number is the greater priority!
- nice command, modifies the priority of the command to be started.
- Default priority is 0, top or ps -l command NI column!
- nice -n 5 sleep 20 & # the new priority will be 5 (it adds 5 to 0)
- nice -n -10 sleep 20 & # only with root permission can give more priority!!!!

Processes, priorities

- Unix and Linux handle processes based on priority!
- POSIX4 –IEEE1003.1b (1993), appears real time extension
- Two priority lists
 - nice -20 – +19, as we saw
 - Real time priority list with values between 0-99. (100 priority levels)
 - In this list the greatest number means the greatest priority
 - The common representation of the two lists is the following:
 - 99,98...1, -20,-19,..0,1,..19 , often it is called a 140 priority interval in which different mappings are possible

Delayed execution of commands

- In Unix systems delayed executions are supported in two different way
 - Cron(demon), superuser permission is needed or to have a permission in/etc/cron.allow
 - /etc/crontab –system timing, /etc/cron.d–user timings
 - These timings decide which program at which time must be started by the cron
 - At – command may be used if atd service is running.
 - At command waits its input from the standard input (the command to run)
 - At now + 5 minutes < commandfile #at needs the time what we can give in a lot of way
 - At 0845 oct 1 <also # also will executed at 8.45 oct 1.

To finish a command

- Normal finishing –the command does not want to run further
- A command runs maximum till that time while it's starting user is logged in!
 - It does not matter whether it is a foreground or background process!
- Nohup command
 - Nohup command & # it may be used without &, but in this case it is a foreground process!
 - The output of the command will be stored in the nohup.out file!
 - This file is created at the moment of starting, after logging out the output will be appended to it!
 - One can define an own output file: nohup command > userdefined.nohup

Apostrophes

- In command line the legal characters are: .,_,-,numbers, normal characters
- Apostrophes: ',",\ – modify the classical meaning of characters
- Among single quotes ' each special influence e.g. special \ \$ % etc. Influence disappears
- E.g.: `echo 'apple $tree' #apple $fa`
- Among double quotes " the \$, a \, a ` and the ' character influence remains
 - E.g.: `a=RealMadrid; echo "Let's go $a!" # Let's go RealMadrid!`
- \x character: modify the original meaning of x
 - E.g.: `tree=flower; echo apple $tree #apple $tree`
 - `Echo apple $tree #apple flower`

Redirection of input and output

- Input, output devices
 - `stdin(0)` -keyboard, standard input channel(default input)
 - `stdout(1)` -monitor, standard output channel(default output)
 - `stderr(2)` –monitor, standard error channel(default error-output)
- Redirection
 - Output: with the help of symbol `>`
 - E.g: `echo apple peach plum > fruit`
 - `Echo apple >&2` #not to be the file named 2
 - Input: with the help of symbol `<`
 - E.g: `passwd steve <apple; cat apple # appletree, appletree`

I/O redirection II.

- Output, append
 - >> filename, e.g.: echo nut >>fruit
 - If it does not exist, then it will be created!
- Error-output(stderr) redirection
 - 2>, 2>>
- For the sake of symmetry(☺): <<
 - Input redirection while not getting the text given after the << (now apple)

```
$ cp 2>myerror  
$ mv this 2>>myerror  
$ cat myerror
```

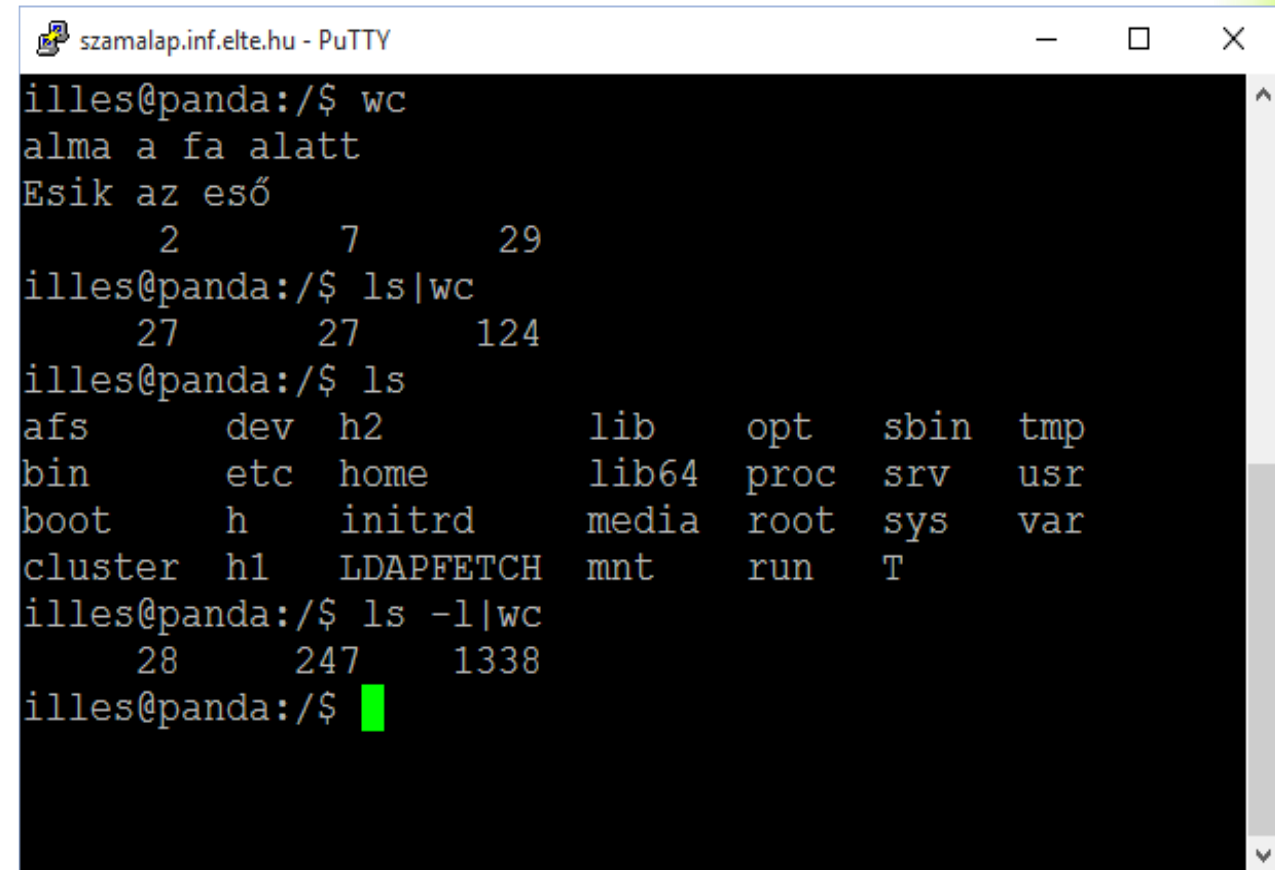
```
cat<<apple  
<input type=text name=X>  
<input type=button>  
apple
```

Filters

- Command vs. filter
 - Its input can be the result of an other command!
- A filter is not a filter in itself!
 - At least it is about the connection of two commands!
- Operator symbol: |
- Such very well known, frequently used command is e.g. wc!
 - Word Count!
 - Task: to count lines, words, characters!
 - `wc [-lwc] [file]` #without a parameter it waits for the input data till CTRL+D!

Filter usage

- May be used as normal commands, without parameters, in this case it waits data from the keyboard!
 - The end of input:CTRL+D
 - Using it with parameters the input may be modified!
 - eg: wc apple.tree
- Filter like, with| character!



The screenshot shows a PuTTY terminal window titled 'szamalap.inf.elte.hu - PuTTY'. The user 'illes@panda' is in the root directory. The terminal shows the following commands and output:

```
illes@panda:/$ wc
alma a fa alatt
Esik az eső
      2      7      29
illes@panda:/$ ls|wc
      27      27      124
illes@panda:/$ ls
afs      dev  h2      lib      opt      sbin  tmp
bin      etc  home    lib64    proc     srv   usr
boot     h    initrd  media    root     sys   var
cluster  h1   LDAPFET mnt      run      T
illes@panda:/$ ls -l|wc
      28      247      1338
illes@panda:/$
```

Important filters

- Important, ready made filters
 - Tee, tr, cut, sort, uniq, wc, grep etc.
 - Eg:

```
$ who >names
```

```
$ sort names #ordered by lines
```

```
$ who | sort -r -u # reverse order, uniquelines
```

```
$ who | wc -l #number of logged in users
```

- Do not forget to ask your man(ual)
 - Eg: man sort, etc.

Tee -, tr commands

- tee – it copies the passing content(through the pipe) into the file(s) given by the parameter(s).
 - E.g.: `ls -l | tee dir.txt | wc -l`, `who | tee -a users.txt report.txt | sort -r`
 - -a parameter: append text to each files
- tr—translate, it has two parameters, two character groups. The characters to be translated arrive to the standard input. If they are given in the first character group tr will replace them with the characters given by the second group.
 - E.g.: `echo append | tr pend rive # arrives`
 - Similar to it the usage of sed y command! (We will see it later!)

Cut – cutting out

- We can cut field(s), column(s) from the lines of the standard input or a file!
- `cut -c1-5` #cut off 1-5 characters
 - E.g.: `date | cut -c4-8` # Oct
- `cut -f1,3,5-7` #cut off 1,3,5,6,7 fields
 - Default delimiter is : Tab
 - New delimiter: -dchar
 - E.g.: `cat/etc/passwd | cut -f1,7 -d:` # name, shell

Grep –filtering lines

- Select lines which are fit to pattern given by the parameter.
- Important parameters:
 - -v lines which do not fit to the given pattern
 - -i not case sensitive
 - -w select it only as a whole word(comPUTer not)
 - -r recursively for the given(parameter) directory.
 - -l Writes out only the file names. (Searches inside the file.)
 - -c Writes out only the number of fitting lines
 - -n Numbering of lines.

Usage of grep

- Filter example:

- `Cat names | grep Steve` # As a result we get the lines which contains Steve.

- Command syntax:

- `grep -r 'Real Madrid' ./script` # it searches for lines containing Real Madrid in the files of directory script
 - `grep -r -l par *` # In each file, in each subdirectory it searches for word par. As a result it writes only the file names.

Patterns, regular expressions

- Defining a (text)pattern – Regular expressions - special characters
 - ^ the pattern must fit from the start of the line.
 - E.g.: '^apple' : apple word is at the beginning of the line.
 - \$ the pattern must fit to the end of the line.
 - E.g.: 'peach\$' : peach word is at the end of the line
 - . Any character
 - * The repetition of the previous pattern 0 or more times!
 - E.g.: '^apple.*tree\$' -between apple and tree may be any number of character(0 also)

Samples I.

- `cat file | grep „^$”` #Empty line
 - How many empty line in a file? `cat file | grep „^$” | wc -l`
- `cat file | grep „^$ FTC.*\$$”` # At the beginning and end position \$ char, after first \$ FTC, and any char!
- `cat file | grep „/-”` # Anywhere /- chars.
- `cat file | grep „- /”` # Error, the – char is a command option prefix, and there is no / option!
- `cat file | grep „\ - /”` # This is OK, -/ chars anywhere!

Other patterns

- To give character sets: [..]
 - [a-z] (small letters)
 - [^a-c] Not an a,b, or c
 - [A-Za-z0-9] Alphanumerical (number or letter)
 - \w Alphanumerical, just the same as the previous
 - \W Not alphanumerical
 - \d digit, the same as [0-9]
 - \s space, tab, line break
- Fitting words
 - \< word start, e.g.: grep "\<Zol"
 - \> word end, e.g.: grep "\>tán"
 - \b Space at begin and end Pl: grep '\bpista\b'

Samples II.

- `cat file | grep [-a]` # The – or a char search
 - `cat file | grep [a-]` # Same. The –char could be at first or last place!
 - `cat file | grep [a-c]` # The a-c interval, soa orb orc chars
 - `cat file | grep [-ac]` # The –ora orc chars
- `cat file | grep [FTC]{})` # F or T or C and after {)}) chars
- `cat file | grep [FTC\]{})` # Inside [] chars there is no spec. meaning, eg. \d or.
- `cat file | grep []FTC{)}` # Or] or other chars...

E(F)Grep–pattern fitting I.

- `egrep`, `fgrep` – extended `grep` -E, fixed, `grep-F`
 - `ali|eva` , or relation,
 - E.g.: `ls | egrep "par|example"`
 - `+` Previous pattern repeats at least once
 - E.g.: `ls | egrep "\<p+f" #` at the beginning of the word 1 or more `p`, then an `f` letter.
 - `?` Previous pattern repeats 0 or once.
 - E.g.: `ls | egrep „param\d?" #` `param`, `param1`, `param2`,...
 - `{n}` previous character punctually `n` times!
 - E.g.: `cat appletree | egrep ^[a-b]\w{5}`

E(F)Grep – Pattern fitting II.

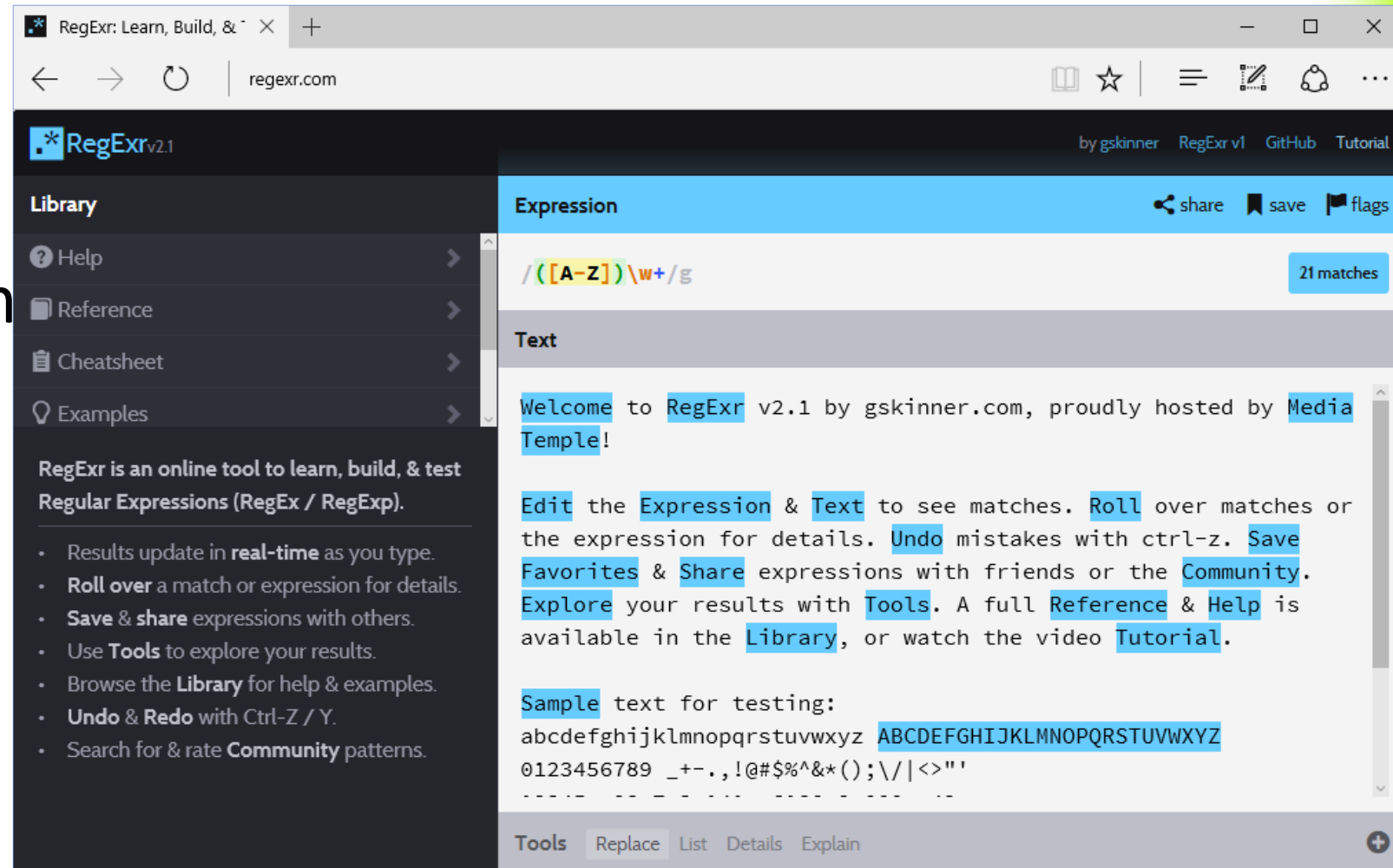
- `{2,4}` Previous pattern repeats 2,3 or 4 times
 - `-{1,}` Previous pattern at least once.
- `()` Makeing a group from a pattern.
 - – It is useful for implementing repetitions. Usually it is followed by a command of repetition `+,?,{n}`
 - –E.g: `[0-9]{8}(\s[0-9]{8}){1,2} #bank account`
 - Instead of `[0-9] \d` would be good as well.
- Special character is practical to precede by a `\` character.
 - –E.g.: `^[+-]? \d+([,\.] \d+)? #a signed number with decimal point or coma`
- For more examples see man!

Other patterns

- E-mail address (shortversion):
 - `\w+[\.-_\w*]*@\w+(\.\w+)+`
- Hour:minute:second
 - `[01][0-9]|2[0-3]:[0-5][0-9]:[0-5][0-9]`
- Etc.
- Each programming language contains this or the extended version of it!

Regular expressions - online

- You can find online sites
- <http://regexr.com>
- <http://regex101.com>
- Etc..



Text editors (VI or VIM) – II.

- Some vi(m) feature!
 - Two working mode, command or editor mode!
 - To change from command to editor mode: i, a, o, etc.
 - To change from editor to command mode: ESC
 - In command mode: saving: w name, save and exit: wq(quit)
 - :help(see picture)
 - Exit from help screen: q
 - Exit without saving: q!

```

szamalap.inf.elte.hu - PuTTY
*help.txt*      For Vim version 7.4.  Last change: 2012 Dec 06

          VIM - main help file

Move around:  Use the cursor keys, or "h" to go left,          k
              "j" to go down, "k" to go up, "l" to go right.      h  l
              "j" to go down, "k" to go up, "l" to go right.      j

Close this window: Use ":q<Enter>".
Get out of Vim:   Use ":qa!<Enter>" (careful, all changes are lost!).

Jump to a subject: Position the cursor on a tag (e.g. |bars|) and hit CTRL-].
With the mouse:   ":set mouse=a" to enable the mouse (in xterm or GUI).
                  Double-click the left mouse button on a tag, e.g. |bars|.

Jump back:       Type CTRL-T or CTRL-O (repeat to go further back).

Get specific help: It is possible to go directly to whatever you want help
                   on, by giving an argument to the |:help| command.
                   It is possible to further specify the context:

                                     *help-context*
                                     WHAT          PREPEND    EXAMPLE
                                     Normal mode command (nothing) :help x

help.txt [Help] [RO]                                     1,1          Top

[No Name] [+]                                           1,0-1        Top
"help.txt" [readonly] 221L, 8249C
  
```


VI(M) – unnamed- named buffer

- Cut – Copy – Paste

- Cut : dd, delete(cut) actual line, the content will be in unnamed buffer
- Copy : yy, copy actual line into unnamed buffer
- Paste : p, copy(insert) the buffer content
- Cut, copy more than one lines: 5dd or 4yy, cut 5 lines, copy 4 four lines into buffer.

- Named buffer

- We can use only one char long buffer names, after double quote : e.g. "a
- "s2yy # Copy the actual 2 lines into S buffer
- "sp # Paste the s buffer content

Thank you!

