Outline

- Introduction - history
- Command line basics – getting help
- File system
- Working with files and directories
- More file handling
- The shell revisited
- ➢Monitoring resources

Check disk space

Measuring disk usage

- `$ du -h <file>`
  -h: returns size on disk of the given file, in human readable format: K (kilobytes), M (megabytes) or G (gigabytes),
  Without -h, du returns the raw number of disk blocks used by the file (hard to read).
  Note that the -h option only exists in GNU du.
- `$ du -sh <dir>`
  -s: returns the sum of disk usage of all the files in the given directory.

Measuring disk space

- `$ df -h <dir>`
  Returns disk usage and free space for the filesystem containing the given directory.
  Similarly, the -h option only exists in GNU df.
- Example:
  `$ df -h .`

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/hda5	9.2G	7.1G	1.8G	81%	/

- `$ df -h`
  Returns disk space information for all filesystems available in the system. When errors happen, useful to look for full filesystems.

How much space do I have?

- `quota`: command to see all quotas for your directories are, if any

```
3.10.0-957.27.2.el7.x86_64
bash-4.2$ echo $SHELL
/bin/bash
bash-4.2$ ls
Desktop                     Videos                mytest.m
Documents                   core.23219            openmp
Downloads                   inbox                 output.txt
MATLABDesktopCreateError.log intel                pbsnodes-list
Matlab_and_Worker_p2.pdf    java.log.14915        result10hpc.txt
Music                       matlab                simple_script_1
Pictures                    matlabtest            test
Public                      mex_sum_openmp.c      test_mex_openmp.m
Templates                   mex_sum_openmp.mexa64 testwo
bash-4.2$ pwd
/vsc-hard-mounts/leuven-user/300/vsc30051
bash-4.2$ quota
quota: error while getting quota from nfshome.usr.hydra.brussel.vsc:/apps/brussel for vsc30051 (id 2530051): Operatio
quota: error while getting quota from nfsdata.usr.hydra.brussel.vsc:/data/brussel for vsc30051 (id 2530051): Operatio
quota: error while getting quota from nfshome.usr.hydra.brussel.vsc:/user/brussel for vsc30051 (id 2530051): Operatio
quota: error while getting quota from nfsdata.gastly.gent.vsc:/user/data/gent for vsc30051 (id 2530051): Operation no
quota: error while getting quota from nfsapps.gastly.gent.vsc:/apps/gent for vsc30051 (id 2530051): Operation not per
quota: error while getting quota from nfshome.gastly.gent.vsc:/user/home/gent for vsc30051 (id 2530051): Operation no
Disk quotas for user vsc30051 (uid 2530051):
     Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
10.118.240.67:/user
                2650468  2831156 3145728          61675   90000  100000
10.118.240.67:/data
                34919020  76546048 78643200           4930  9000000 10000000
bash-4.2$
```

Process management

Process

- Start a process (= run a command), 2 ways to execute
  - Foreground Processes
    - depend on the user for input
    - also referred to as interactive processes
  - Background Processes
    - run independently of the user
    - referred to as non-interactive or automatic processes
    - Daemons: special type of background processes that start at system startup and keep running forever as a service; they don't die.

&

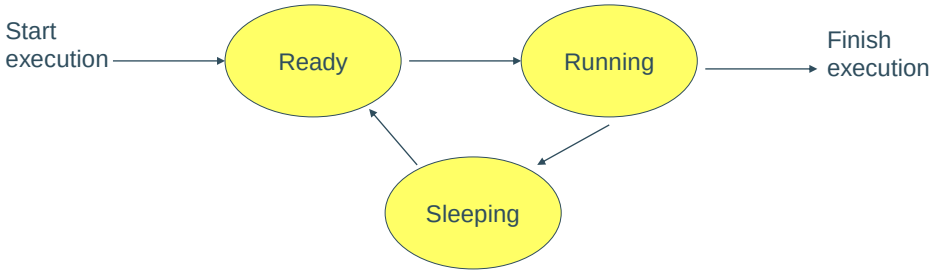
- **&** is a command line operator that instructs the shell to start the specified program in the background.
  - This allows you to have more than one program running at the same time without having to start multiple terminal sessions.
  - Starting a process in background: add & at the end of your line:
    `$ nano &`
    check with `ps`
  - Put process in background with ctrl-z
  - Bring a process to the foreground: `fg` (followed by the background process number)
  - Check with: `jobs`

Process

- Processes carry out tasks within the operating system.
- Several instances of the same program can run at the same time
- Processes are assigned a unique identifier which is used to monitor and control the process (PID)

Process

- A program that is claimed to be executing is called a process
- For a multitasking system, a process has at least the following three states:



Process

- Ready state
  - All processes that are ready to execute but without the CPU are at the ready state
  - If there is only 1 CPU in the system, all processes except one are at the ready state
- Running state
  - The process that actually possesses the CPU is at the running state
  - If there is only 1 CPU in the system, there is only one process that is at the running state
- Sleeping state
  - The process that is waiting for other resources, e.g. I/O

ps

- **p**rocess **s**tatus: display running processes
  (cfr. Windows Task Manager ctrl-shift-esc)
- `$ps` Display the current user's processes
- `$ps -e` Display all processes running on the system
- `$ps -ef` Display detailed information about running processes
- `$ps -u [USER]` Display processes owned by the specified user
- `$ps -aux`

  a = show processes for all users
  u = display the process's user/owner
  x = also show processes not attached to a terminal

ps

```
  PID   TTY     STAT   TIME COMMAND
14748   pts/1   S      0:00 -bash
14795   pts/0   S      0:00 -bash
14974   pts/0   S      0:00 vi test1.txt
14876   pts/1   R      0:00 ps …
```

Process ID

Controlling Terminal name

State:
S – Sleeping (waiting for input)
R – Running

Total CPU usage

Name of executable/command

- For the example above, both bash processes, which are the shell of both terminals, are waiting for the input of user. They must be in the sleeping state
- The vi process, which is an editor, is also waiting for the input of user. Hence it is also in sleeping state
- When ps reporting the processes in the system, it is the only process that is running. Hence it is in running state

ps -aux

```
(base) frankvp@CRD-L-08004:~$ ps -aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   1520  1132 ?        Sl   07:52   0:00 /init
root        19  0.0  0.0   1184   360 ?        Ss   08:25   0:00 /init
root        20  0.0  0.0   1184   368 ?        R    08:25   0:00 /init
frankvp     21  0.0  0.0  10832  6036 pts/0    Ss   08:25   0:01 -bash
root       170  0.0  0.0   1184   360 ?        Ss   11:54   0:00 /init
root       171  0.0  0.0   1184   368 ?        S    11:54   0:00 /init
frankvp    172  0.0  0.0  10188  5140 pts/1    Ss+  11:54   0:00 -bash
frankvp    371  0.0  0.0   8364  3204 pts/0    T    15:49   0:00 nano
frankvp    372  0.0  0.0   8624  3200 pts/0    S    15:50   0:00 /bin/bash ./run_hello_world.sh
frankvp    373  0.0  0.0   7236   584 pts/0    S    15:50   0:00 sleep 10
frankvp    374  0.0  0.0  10860  3352 pts/0    R+   15:50   0:00 ps -aux
(base) frankvp@CRD-L-08004:~$
```

- More fields:

- USER: The effective user (the one whose access we are using)
- PID: Process ID
- %CPU: CPU time used divided by the time the process has been running
- %MEM: Ratio of the process's resident set size to the physical memory on the machine
- VSZ: Virtual memory usage of the entire process
- RSS: Resident set size, the non-swapped physical memory that a task has used
- TTY: Controlling terminal associated with the process
- STAT: Process status code
- START: Start time of the process
- TIME: Total CPU usage time
- COMMAND: Name of executable/command

Process state codes

- R: running or runnable (waiting for the CPU to process it)
- S: Interruptible sleep, waiting for an event to complete, such as input from the terminal
- D: Uninterruptible sleep, processes that cannot be killed or interrupted with a signal, usually to make them go away you have to reboot or fix the issue
- Z: Zombie, are terminated processes that are waiting to have their statuses collected
- T: Stopped, a process that has been suspended/stopped

kill

- Sends an abort signal to the given processes. Lets processes save data and exit by themselves. Should be used first.
- `$ kill <pid>`
  Example:
  `$ kill 3039 3134 3190 3416`
- `$ kill -9 <pid>`
  Sends an immediate termination signal. The system itself terminates the processes. Useful when a process is really stuck.

killall

- The `killall` command terminates all processes that match the specified name
- `$ killall [-<signal>] <command>`
  Example:
  `$ killall bash`

top

- Displays most important processes, sorted by cpu percentage
- To sort by other fields press < to move the sort column to the left and > to move the sort column to the right

  http://www.thegeekstuff.com/2010/01/15-practical-unix-linux-top-command-examples/