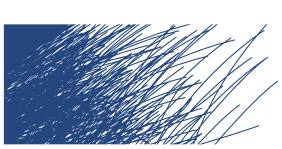
Introduction into Linux, Bash & HPC

Ward Poelmans

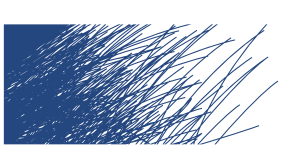
Ghent University

14 september 2016




UNIVERSITEIT GENT

Contents




UNIVERSITEIT GENT

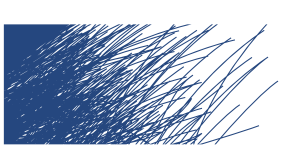
What is Linux?

- Operation System
- Started as a hobby project by Linus Torvalds in 1991
- Based on the UNIX philosophy
- Open Source!
- Many independent components: kernel, glibc, bash, X11, ...
- Runs on a large part of the internet infrastructure. Android is Linux based.
- The defacto standard in the HPC world!

UNIVERSITEIT GENT

Linux Distributions

A more or less complete list of distros




UNIVERSITEIT GENT

What is Bash?

Several ways to interact with Linux:

- Graphically: X11, Wayland, Android, ... $\Rightarrow$ Gnome, KDE, Unity, ...
- Text: commandline interface

A commandline interface needs a commandline processor: Bash, Korn, C-shell, Bourne, Busybox, ...

- Gives you a prompt to enter commands
- Allows to combine multiple commands
- You can write scripts in them
- ...

UNIVERSITEIT GENT

Connect to a Linux machine

On Windows:
Start Putty, enter the name of the machine (molgate.ugent.be) and the port 45917. Next, enter username and password.

On Linux:
Open a terminal, enter
`ssh -X -p 45917 <username>@molgate.ugent.be` and then your password.

On Mac OS X:
Start Terminal.app and enter
`ssh -X -p 45917 <username>@molgate.ugent.be` and then your password.

UNIVERSITEIT GENT

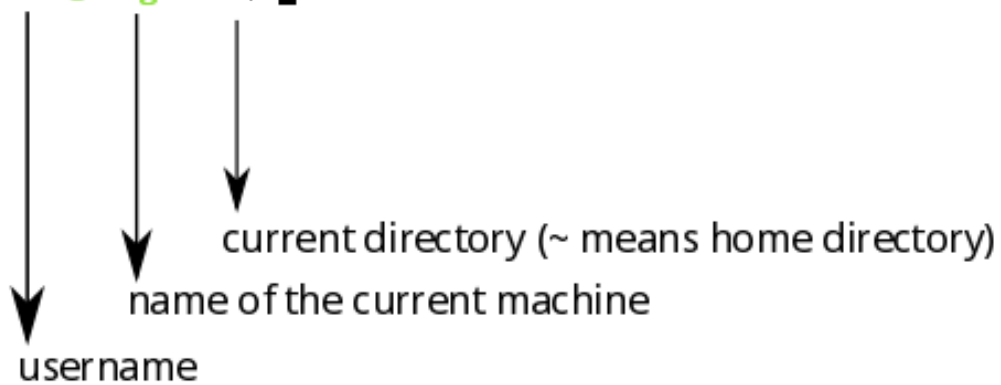
Bash Prompt

File Edit View Search Terminal Help

```
ward@molmod57 ~ $ ssh molgate
Linux molgate 3.2.0-4-amd64 #1 SMP Debian 3.2.46-1 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Sep 16 15:56:35 2013 from molmod57.ugent.be
ward@molgate:~$
```



UNIVERSITEIT GENT

Some commands

- `ls` shows the files in the current directory
- `id` tells who you are and in what groups you are.
- `cd` allows you to change the current directory
- `pwd` tells you the current directory

Try `ls -alh`
This means, run the `ls` command with the options `-a` (all), `-l` (list) and `-h` (human readable)
Bash keeps a history. Use the arrows keys to navigate through it.

UNIVERSITEIT GENT

ls command

```
File  Edit  View  Search  Terminal  Help
ward@molgate:~$ ls -alh
total 164K
drwx------  6 ward cmm  4.0K sep 16 16:58 .
drwxr-xr-x 18 root root 4.0K sep 16 16:40 ..
-rw-r--r--  1 ward cmm   111 nov 11  2010 .bash_aliases
-rw-------  1 ward cmm  6.1K sep 16 16:33 .bash_history
-rw-r--r--  1 ward cmm   220 nov 11  2010 .bash_logout
-rw-r--r--  1 ward cmm  3.1K nov 11  2010 .bashrc
drwxr-xr-x  2 ward cmm  4.0K sep 16 16:32 .fontconfig
-rw-r--r--  1 ward cmm    23 sep 21  2011 .forward
-rw-------  1 ward cmm  2.1K mei  4  2012 .gnuplot_history
-rw-r--r--  1 ward cmm    49 apr 20  2012 .gnuplot-wxt
-rw-r--r--  1 ward cmm  2.7K sep 16 16:58 klotski.hs
-rw-r--r--  1 ward cmm  2.7K feb 13  2011 klotski.hs~
-rw-------  1 ward cmm    58 feb 22  2013 .lesshst
drwxr-xr-x  7 ward cmm  4.0K aug 30 13:54 linmol
-rw-r--r--  1 ward cmm   10K jun 24  2012 mailfiles.tar
drwxr-xr-x  2 ward cmm  4.0K nov 21  2011 .ncftp
-rw-r--r--  1 ward cmm  1.7K jan 21  2011 preamble.tex
-rw-r--r--  1 ward cmm   675 nov  5  2010 .profile
-rw-r--r--  1 ward cmm   22K dec  7  2010 punten.ods
drwx------  2 ward cmm  4.0K jun 22 14:44 .ssh
-rw-r--r--  1 ward cmm   24K apr 16 19:37 Verslag OCTN 20130415.odt
-rw-------  1 ward cmm   11K sep 16 16:58 .viminfo
-rw-r-----  1 ward cmm   11K nov 11  2010 .vimrc
-rw-------  1 ward cmm    53 sep 16 16:33 .Xauthority
ward@molgate:~$
```

size of files in current directory

name of the file

time of last change

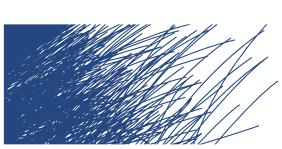
size

group

owner

permissions

type




UNIVERSITEIT GENT

Permissions

Every file and directory has a owner and a group and permissions associated with them. There are 3 kind of permissions:

- read (r): you can read the contents of the file
- write (w): you can write to the file
- execute (x): you can execute the file (if it's a program), also needed to open a directory

Some examples:

```
drwxr-xr-x 7 root root 4.0K aug 30 13:54 linmol
-rw-r----- 1 ward cmm  22K dec  7  2010 punten.ods
```

`root` is the superuser (aka God)

UNIVERSITEIT GENT

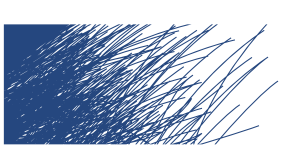
Jumping around

The `cd` commands allows you to change the current directory.

- `cd` jump to your home directory
- `cd <name of directory>` jump to a directory
- `cd ..` jump to the directory one level higher

When you need to specify a filename to a command, use the TAB key for autocompletion: enter the beginning of the filename and press TAB to autocomplete. Press TAB twice to see all possibilities.
In Linux, the directory structure is a tree with one root (case sensitive):

```
/                          /dev
/home/<username>           /proc
/usr/bin                   /sys
/usr/lib                   /boot
/tmp                       /var
```

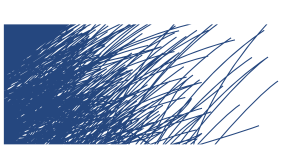



Execute commands

General form: `<name of the program> <options>`
How to find out the possible options?

- `<name of the program> --help`
- `man <name of the program>` (press q to exit)

Often used options have a long (`--help`) and a short form (`-h`)

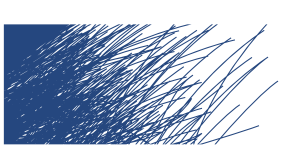



UNIVERSITEIT GENT

Working with files

- `mkdir` creates a directory
- `mv` moves a file or directory
- `cp` copies a file or directory
- `rm` deletes a file or directory (permanently)

Often used options: `cp -a` tries to make an identical copy (same permissions, date, ...). `rm -rf` removes recursively and deletes directories.



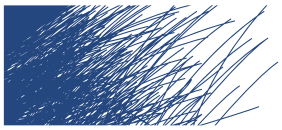

UNIVERSITEIT GENT

Copying files between PC's

- `scp` copies files between 2 Linux machines
- `winscp` does the same between Windows and Linux
- `rsync` intelligently copies between 2 Linux machines

Examples: `scp -r <local dir> <name_of_machine>:<remote dir>`
`rsync -av <local dir> <name_of_machine>:<remote dir>`

For `rsync`, the first directory name is special:
`rsync -av map somemachine:` means copy 'map' to somemachine.
`rsync -av map/ somemachine:` means copy the contents 'map' to somemachine.




UNIVERSITEIT GENT

Editors

Many editors:

- emacs: highly extensibility, has buildin webbrowser, ...
- vim: arch-nemesis of emacs, very powerfull but steep learn curve
- nano: very easy in use

To open a file, you give it as an argument of the program:

```
nano bestand.txt
```

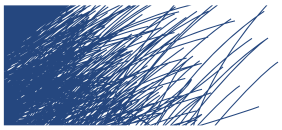
UNIVERSITEIT GENT

Let's try something!

`less` and `cat` allow you to view to contents of a file.

Open the file `/usr/share/doc/bash/copyright` with `less`.
Open the file `/usr/share/doc/bash/README` with `cat`. Show the contents of the directory. Find out the permissions of the `FAQ` file.

Change permissions: `chmod`
`chmod ugo±rwx <file>`
Example: `chmod u+r` add the read permissions for the owner. Check the permissions of your home directory and secure them (only you can read and write to it). Create a directory in your home directory and copy the `README` and `copyright` file to it. Add a line to `README` file. Use `winscp` to copy the directory to your Windows PC. Afterwards, delete the directory.

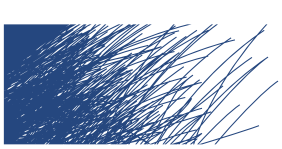



UNIVERSITEIT GENT

Input-Output redirecting

UNIX philosophy: use many small programs
In bash, the output of one program can serve as the input of another.
Example: `cat <file> | wc -c`
The | redirect the output of the program before the | to the input of the program behind the |

Try counting the number of files in the `/usr/bin` directory. (tip: use `wc`)



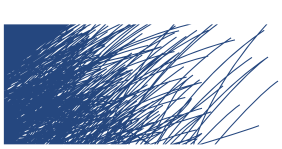

UNIVERSITEIT GENT

Output to file

Every program has 2 outputs:

- Standard Output (send through the pipe)
- Standard Error Output (not send through the pipe)

To send the output of a program to a file, you can $>$.
Example: `cat <file> > file2`
Redirect Standard Error Output:
`cat <file> 2> file3`
Redirect both Standard Output and Standard Error Output:
`cat <file> &> file3`

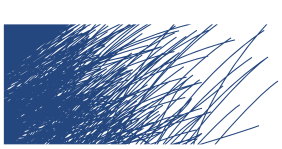



UNIVERSITEIT GENT

Let's use programs!

Wildcard: `*` means anything. For example: `ls CMM*.txt`

- `find` can find stuff
  Examples: `find /some/dir -name 'name_of_file'`
- `sort` sorts stuff (use `-n` to sort numerically)
- `grep` allows you to 'grep' in a file
- `sed` allows you to replace things in a file
- `xargs` allows you to do the same thing on many files
- `cut` can cut a column out of a file
- `head` show the head of a file
- `tail` show the tail of a file (with `-f` it will follow the file for any adds)




UNIVERSITEIT GENT

Regular Expressions

Regular Expressions allow you to match a pattern. Extremely powerful! Examples:

- `.` matches any character
- `[a-z]` matches any letter in the range a-z. Add a `^` to invert.
- `^` and `$` match the beginning and the end of a line
- `*` tells that the previous pattern should be matched zero or more times
- `+` tells that the previous pattern should be matched one or more times
- `?` tells that the previous pattern should be matched zero or one time
- Patterns can be grouped with `( )`
- `|` means a OR operation between the patterns before and after it

Regular Expressions

Some pratical examples:

- `^.*Energy: [0-9.]*$` matches any line that ends with word Energy followed by a number
- `(CMM|UGent)` matches any line with the word CMM or UGent on it
- `\bCMM\b` matches the word CMM

`grep` and `sed` understand these. Knowing regular expressions can make your life easier...

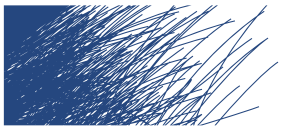
UNIVERSITEIT GENT

Let's do something!

First do: `wget -O exercises.tar.gz http://goo.gl/4T0R6D`
Then: `tar -xvzf exercises.tar.gz` (this will unpack the archive in the current dir)

Grep the optimized energy out of the file '`H20.log`', grep the optimized geometry

Tips: `grep -A`

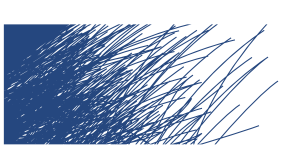



UNIVERSITEIT GENT

Let's do something!

The file '`cp2k.ener`' contains several columns with data. Find the maximum and the minimum temperature. Tip: sort

Make a new file with only the time and the kinetic energy.
Tips: `cut`, more advanced: `awk`
`cut -f2-3` cuts columns 2 and 3 out of a file. The columns are separated by a single tab.




UNIVERSITEIT GENT

awk

`awk` is a powerful data processing language: very quick introduction.

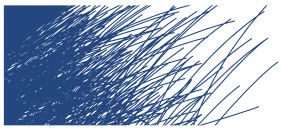
`awk '{ print $1 }'` will print column one of a file. A column is separated by spaces or tabs.

`awk '/^[^#]/ { print $1,$2,$2+$1 }'` will only print lines that don't start with a '#'. It prints column 1, 2 and the sum of 1 and 2

`awk -F: '{ print $1,$2 }'` changes the seperator between columns to a ':'

`awk 'BEGIN { sum = 0 } /^[^#]/ { sum += $2 } END { print sum }'` sums all values in column 2.

Calculate the average temperature. Python can do everything `awk` can and more but `awk` is easier on the commandline.

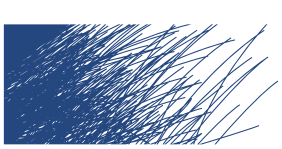



UNIVERSITEIT GENT

Time for some scripting

A script is a list of commands to bash. Useful to automate things. Needed for the HPC. To make a file a bash script, it must start with the sha-bang: `#!/bin/bash` Further, it needs to be executable (+x). You can run it with `./<filename>`

Bash has also a lot of programming abilities: loops, variables, ...




UNIVERSITEIT GENT

Bash Programming 101

Variables are called environment variables. `set` gives a list of the current variables. You can access them with the $ sign. Examples:
`echo $HOME ; a="$HOME/blabla" ; b='$HOME' ; echo "$a is not $b"`
A variable is local. You have to export it to make visible to other processes: `export a` Use `source <file>` to process all statements in the current shell.

With `$( ... )` or ` ... ` you can start a subshell. Example:
`wie=$(id | cut -f1 -d ' ') ; echo $wie`

Tier2 @ UGent: STEVIN





	raichu	delcatty	phanpy	golett	swalot
# nodes	64	158	16	200	128
# cores	1024	2528	384	4800	2560
Interconnect	Ethernet	Infiniband FDR	Infiniband QDR	Infiniband QDR	Infiniband FDR
CPU	Intel Xeon Sandy Bridge	Intel Xeon Sandy Bridge	Intel Xeon Haswell	Intel Xeon Haswell	Intel Xeon Haswell
Clock (GHz)	2.6	2.6	2.5	2.5	2.6
Memory per node (GiB)	32	64	512	64	128
Installed	2012	2013	2015	2015	2016

Picking a cluster

BATCH clusters: single core/single node jobs

- no fast interconnection network
- no shared scratch attached, only local disk
- use *$VSC_SCRATCH_NODE*, *$TMPDIR*

MPI clusters: multi-node jobs

- fast Infiniband interconnect
- dedicated shared scratch storage (fast)
- use $VSC_SCRATCH_<NAME>

Picking a cluster

High memory jobs

- intended for single-node jobs
- large amount of memory available

Tier1

- usage is not free
- access is project based
- see: *https://www.vscentrum.be/en/access-and-infrastructure/tier-1-clusters*

GPU

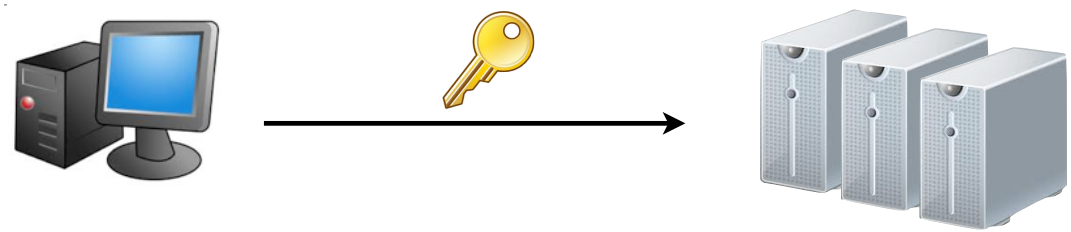
- not available in UGent
- reachable at KULeuven (free of charge)
- see: *http://hpc.ugent.be/userwiki/index.php/Tips:Software:GPGPU*

Logging in

Accessing STEVIN is via an **SSH connection** to login nodes:

```
ssh vsc40000@login.hpc.ugent.be
```

Windows: using PuTTY, Mac OS X: using Terminal.app



Authentication is done using **public/private key pair**

use a password to protect your private key!

Login nodes are named *gligar0[1-3]*:

```
$ hostname
gligar02.gligar.os
```

Submitting and managing jobs

Include reasonable PBS directives in your job script:

```
#!/bin/bash
#PBS -N solving_42  ## job name
#PBS -q default  ## default queue (~ requested resources)
#PBS -l nodes=1:ppn=all  ## single-node job
#PBS -l walltime=10:00:00  ## max. 10h of wall time
#PBS -l vmem=4gb  ## max. 4GB virtual memory

<rest of job script>
```

Submitting and managing jobs

Use the available environment variables:

- **`$PBS_O_WORKDIR`**
  directory in which job was submitted
  e.g., use `cd $PBS_O_WORKDIR` on top
- **`$PBS_JOBID`**
  job id of running job
- **`$PBS_ARRAYID`**
  array id of running job
  only relevant when submitting array jobs (`qsub -t`)
- **`$EBROOTFOO`**, **`$EBVERSIONFOO`**
  root directory/version for software package Foo,
  only available when module is loaded
- different filesystems: **`$VSC_HOME`**, **`$VSC_DATA`**, **`$VSC_SCRATCH`**, **`$VSC_SCRATCH_NODE`**, **`$TMPDIR`**

HPC-UGent infrastructure: **Tier2** (STEVIN)

Overview of different filesystems

- **HOME** (`$VSC_HOME`):
  - slow access, low volume (max. 3GB)
  - for a limited number of small files (e.g., scripts)
- **DATA** (`$VSC_DATA, $VSC_DATA_VO`):
  - for 'long-term' storage of common data
  - slow access (especially non-streaming)
  - for large volumes of data
- **SCRATCH** (`$VSC_SCRATCH`, `$VSC_SCRATCH_VO`, ...):
  - working directory for (multi-node) jobs
  - beware on non-MPI clusters! (gastly, haunter, raichu, ...)
- **LOCAL DISK** (`$VSC_SCRATCH_NODE`, `$TMPDIR`):
  - working directory for single-node jobs
  - should be preferred on non-MPI clusters, if volume permits
  - `$TMPDIR`: unique directory on local disk (e.g., `/local/$PBS_JOBID`)

NO BACKUPS!!!

see also ***http://hpc.ugent.be/userwiki/index.php/User:StorageDetails***

Submitting and managing jobs

Include reasonable PBS directives in your job script:

```
#!/bin/bash
#PBS -N solving_42  ## job name
#PBS -q default  ## default queue (~ requested resources)
#PBS -l nodes=1:ppn=all  ## single-node job
#PBS -l walltime=10:00:00  ## max. 10h of wall time
#PBS -l vmem=4gb  ## max. 4GB virtual memory

<rest of job script>
```

Submitting and managing jobs

Simulations, experiments, ... are written as **job scripts**.

Submit job scripts to a cluster for execution using `qsub`:

```
$ module swap cluster/raichu
$ qsub job.sh
123456789.master13.raichu.gent.vsc
```

An **overview** of the active jobs is available via `qstat`:

```
$ qstat
Job id            Name    User        Time Use S Queue
----------------- ------ ---------- -------- - -----
47496.master13    job1    vsc40000     045:39: R long
47497.master13    job2    vsc40000     050:58: R long
```

To **remove** a job that is no longer necessary, use `qdel`:

```
$ qdel 123456789
```

Think before you act!

Submitting and managing jobs

The **scheduler** decides which job will start next.

All our clusters use a **fair-share scheduling** policy.

No guarantees on when job will start, so **plan ahead**!

Job **priority** is determined by:

- **historical usage**
  - aim is to balance usage over users
  - infrequent users get higher priority
  - (recent) frequent users get lower priority
- **requested resources** (# nodes/cores, walltime, memory, ...)
  - high resource demand => lower priority
- **time waiting** in queue
  - queued jobs get higher priority over time
- **user limits**
  - avoid that a single user fills up an entire cluster

Working with modules

All user-end software is made available via **modules**.

Modules prepare the environment for using the software.

Module **naming scheme**:

```
<name>/<version>-<toolchain>-<suffix>
```

Load a module to use the software:

```
module load Python/2.7.3-ictce-4.0.6
```

See **currently loaded** modules using:

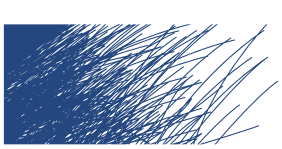
```
module list
```

Get overview of **available** modules using:

```
module avail
```

Only mix modules built with the **same compiler toolchain**.

e.g., ictce (Intel compilers, Intel MPI, Intel MKL (BLAS, LAPACK))




UNIVERSITEIT GENT

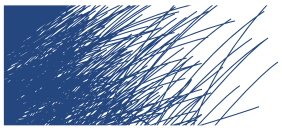
Bash Programming 101

for loop:

```
for I in $(seq 1 2 10); do
  echo "I = $I"
done
```

Read a file line by line:

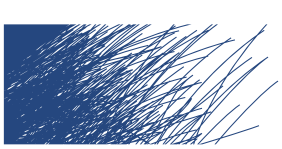
```
cat <file> | while read lijn; do
  echo $lijn
done
```




UNIVERSITEIT GENT

Let's try it!

In the directory `vasp` there are a bunch of files. Extract the energy and the volume and put it into 2 columns.




UNIVERSITEIT GENT

Basic Math

Bash can do integer calculations: `a=1 ; b=$(( 2*a+1 ))`

For floating point, use `bc`: `echo "1/2+3" | bc -q -l`

Make a function table for the function $3x^2 + 9 * x - 2$ and put it in a file. Try using a while loop.

Basic Math

Bash can do integer calculations: `a=1 ; b=$(( 2*a+1 ))`

For floating point, use `bc`: `echo "1/2+3" | bc -q -l`

Make a function table for the function $3x^2 + 9 * x - 2$ and put it in a file. Try using a while loop.

`gnuplot` is a plotting program. Plot the function, based on the function table. The command is:

```
plot 'file' using 1:2 with lines
```

bash tests

Bash has a test function. The syntax is `[[ ... ]]` Possible tests are

- `[[ -a <file> ]]` true is `<file>` exists
- `[[ -d <dir> ]]` true is dir is a directory
- `[[ -z <string> ]]` true if length of string is zero
- `[[ <string1> == <string2> ]]` true if strings are equal
- `(( number1 < number2 ))` the usual numerical comparison.
- `||` and `&&` are OR and AND operations

An if construct is formed like:

```
if [[ ]]; then
  echo 'true'
elif [[ ]]; then
  echo 'something true'
else
  echo 'false'
fi
```

UNIVERSITEIT GENT

Let's try it!

In the directory '`Hubbard`', there are 15 dir with a file '`output.txt`'.
Extract the energy and the value of U out of each file, if it has converged. Sort them and put it them in a file.
Tip: use a loop. grep, cut are your friends.
The last number in the file is the number of iterations: add this in a third column.

sed

`sed` is powerful program, mainly used for search and replace commands. Syntax: `sed 's/search text/replace text/options'`. Possible options:

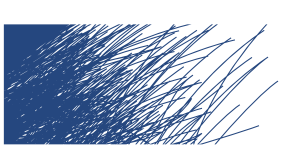
- `I`: case insensitive
- `g`: replace all (by default it only replaces the first occurence on a line
- `p`: print the lines where something has been replace, use together with the `-n` option.
- `-i`: option to sed itself, overwrite the current file

Groups are very powerful. Example:

```
sed 's/^Energy: \([0-9.]*\).*/\1/'
```

You can test a line before the search and replace:

```
sed '/Energy/ s/.*\([0-9.]*\).*/\1/'
```



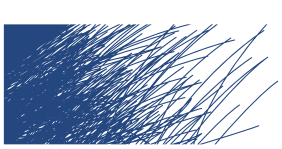

UNIVERSITEIT GENT

find & xargs

`find` does exactly what you expect. Very powerful to select files. Syntax:
`find /some/dir -name '*.txt' -print`
A short overview of options:

- `-name ''` the name of the file/dir must match the pattern
- `-type f|d` tests for file or directory
- `-not -...` invert test
- `-empty` test if empty
- `-depth 1` depth to search for files
- `-newer <file>` true if newer then `<file>`

`xargs` executes command on all files given in the input. Combines very well with `find`

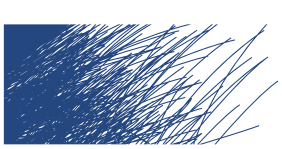



UNIVERSITEIT GENT

find & xargs

`xargs` Syntax:
`find -name '*.txt' -print | xargs -I{} cp {} /some/dir` This will copy all files '*.txt' to a directory. Often used options:

- `-t` show the command that `xargs` executes
- `-L <num>` use at most num input line per command
- `-I{}` replace `{}` with the current input




Let's do something

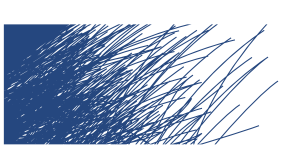
In the directory `HPC`, there is a template jobfile for a calculation. The values for `NNNNNN` and `UUUUUU` still have to be filed in. Generate job files for N=36,34,30,24 and an U range from 1..15.
Submit all those jobs (To submit use the command `stat` instead of the real thing)

UNIVERSITEIT GENT

Let's do something

In the directory `HPC`, there is a template jobfile for a calculation. The values for `NNNNNN` and `UUUUUU` still have to be filed in. Generate job files for N=36,34,30,24 and an U range from 1..15.
Submit all those jobs (To submit use the command `stat` instead of the real thing)

Start to submit jobs, but the haunter cluster is full. You want to submit the remainder to the raichu cluster. Raichu has 16 cores instead of 8. Replace `ppn=8` with `ppn=16` for the N=24 and N=30 jobs.




UNIVERSITEIT GENT

Some useful things

The file `.bashrc` is always executed when you log in.

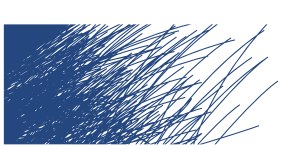
UNIVERSITEIT GENT

Some useful things

The file `.bashrc` is always executed when you log in.

You can provide alias for long commands:

```
alias getdebug='qsub -q debug -I -d $PWD -l walltime=0:59:00
-l nodes=1:ppn=1'
```




UNIVERSITEIT GENT

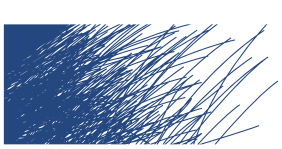
Some useful things

The file `.bashrc` is always executed when you log in.

You can provide alias for long commands:

```
alias getdebug='qsub -q debug -I -d $PWD -l walltime=0:59:00
-l nodes=1:ppn=1'
```

The variable `$?` always has the exit status from the previous program.
Zero means succesful ending. Anything else means an error.




UNIVERSITEIT GENT

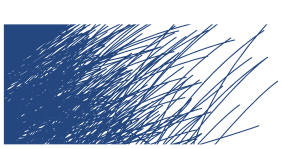
Some useful things

with `ps axuf` you can view the current process. `top` gives a live view and show the cpu and mem usage of each process.

To kill a process there are 2 ways:

- If the process is currently running in your terminal: Control+C
- Use a signal: SIGTERM and SIGKILL

Signals are send with `kill` or `killall`
Example: `killall -TERM bash`




UNIVERSITEIT GENT

The end

Linux and Bash are very powerful but it takes time to master them.

What to do when you need help:

- Use the `--help` option
- Check the man page of the command
- Bash Cheat Sheet
- Google is your friend!
- Ask somebody

To complete your training:
The Advanced Bash scripting guide

HPC-UGent user wiki

Documentation is available at the user wiki:

http://hpc.ugent.be/userwiki

- getting an account
- writing job scripts
- submitting and managing jobs
- working with array jobs
- overview of available software
- tips and tricks
- software-specific documentation (user-editable!)
- creating/joining a virtual organization (VO)
- ...

Contacting HPC-UGent support

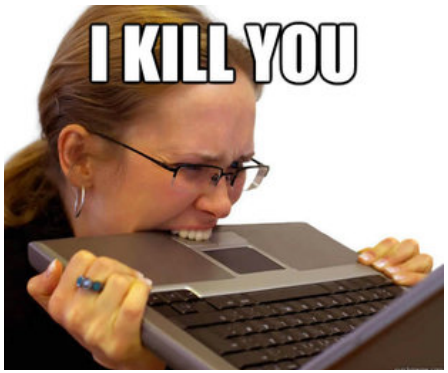
Easiest way to contact HPC team for **support** is via e-mail.

Direct submission into HPC queue of UGent helpdesk:

hpc@ugent.be

Always include:

- clear description of problem (or question)
- location of job script and output/error files in your account
- job IDs, which cluster
- VSC login id
- use your UGent email address, preferably



Alternatives:

- short meeting (for complex problems, big projects)
- hpc-users mailing list (depends on the problem)