Bash Workshop Handout

Accessing remote machines

ssh (secure shell)

ssh tutorialXX@athena.cyfronet.pl	connect to the Athena supercomputer using a tutorial account

Directories

absolute path - starts with “/” and contains all upper directories
e.g.: /net/people/plgrid/tutorialXX/directory

relative path - contains a path starting in the current directory
e.g.: tutorialXX/directory (current directory: /net/people/plgrid)

pwd (print working directory)

pwd	shows an absolute path of current directory, but includes links
pwd -P	shows true absolute path of current directory

Example in the same directory:
pwd -> e.g.: /net/people/plgrid/tutorialXX/scratch
pwd -P ->e.g.: /net/ascratch/people/tutorialXX

cd (change directory)

cd PATH	move to directory specified by PATH, PATH can be absolute or relative
cd ..	move to directory above
cd ../..	move to the directory two lvls above
cd	move to home directory, synonyms: **cd ~**, **cd $HOME**
cd ../PATH	move to a subdirectory of a directory above
cd -	return to last directory visited
cd .	move to the current directory (do nothing)
cd -P .	move to the current directory and remove links from path (**pwd** will print true path from now on)

ls (list)

ls	list files and directories in current directory (you can mix the flags)
ls PATH	list files and directories in a subdirectory
ls -1	**ls** line by line
ls -t	**ls** sorted by date modified, newest first
ls -a	**ls** including hidden files (starting with .)
ls -l	**ls** more information about each element (e.g. size of files in bytes)
ls -h	**ls** normal units (MB etc.)

Arrow up - use the arrow key “up” to use the previous command. Use it multiple times to find older commands. **Arrow down** to return to the newer command.

Ctrl+R - use the “Ctrl” + “R” keys to enter “reverse-i-search” mode. Write a part of a command to find an older command by name.

*** (wildcard)** - use it to replace parts of a name of a file / directory while using other command, cmd will autocomplete it

e.g. cd direct* is going to work as cd directory

? - works like wildcard, but only one character at a time

Tab - use the “Tab” key to autocomplete the name of a file / directory / command while writing it.

--help - write this flag after any command to get info about its flags (works for most commands).

Creating and removing files

mkdir (make directory)

mkdir NAME	create a directory in a current directory named NAME
mkdir -p NAME	**mkdir** but will not return an error when directory NAME exists

touch

touch EXISTING_NAME	update the modification time of a file to now
touch NEW_NAME	create a file name NEW_NAME
touch NAM*	**touch** all existing files starting with NAM

cp (copy)

cp FILE NAME	copy a file and change its name to NAME
cp FILE PATH/.	copy a file to a directory in PATH and don't change its name
cp FILE PATH/NAME	copy a file to a directory in PATH and change its name to NAME
cp -r DIR NAME	copy a directory and change its name to NAME, unless a directory NAME exist, then copy it inside, the result will be: NAME/DIR
cp NAME{,_backup} - will add "_backup" to the copy's name	

mv (move)

mv NAME NNAME	change file's or directory name to NNAME
mv NAME PATH/.	move a file or a directory to PATH and don't change its name
mv NAME PATH/NNAME	move a file or a directory to a directory in PATH and change its name to NNAME

rm (remove, use with caution!)

rm FILE	permanently remove a file in current directory
rm -r DIR	! permanently remove a directory, all its subdirectories and all files inside it
rm -f FILE	! permanently remove a file and **ignore all warnings about it**
rm -rf DIR	!! permanently remove a directory, all its subdirectories and all files inside it **without any warnings**
rm -rf *	<u>!!! permanently remove everything in current directory and all subdirectories</u> - **will ignore all warnings | do not use unless you are sure what you are doing**
rm -rf /*	<u>!!! permanently remove everything on your machine</u> (e.g. your computer)* - **will ignore all warnings | do not use unless you are sure what you are doing**

*Command **rm** will not remove any files that you do not have access to. This means that *this* command will not delete all files from Athena supercomputer, <u>it will exclusively find all your files and remove them.</u>

Ctrl+C - use the "Ctrl" + "C" keys to stop a process. Should you use **rm** by accident, these keys let you stop it.

Ctrl+D - use the "Ctrl" + "D" keys to log out of the machine (Athena). The console must be empty for that to work. Combination **Ctrl+C**, then**Ctrl+D** should force the log out.

Reading files

echo

echo “STRING”	print a string onto console with new line at the end
echo -n “STRING”	**echo** without new line at the end
echo “STRING” > FILE	! replace entire text of a file with one string
echo “STRING” >> FILE	add a string to the end of a file

cat (concatenate)

cat FILE	print contents of a file to console
cat FILE > FILE2	! replace FILE2 with the FILE. If FILE2 does not exist, this creates a copy of the FILE named FILE2
cat FILE >> FILE2	add contents of FILE at the end of FILE2
cat FILE1 FILE2	print two files at the same time

less (print less)

less FILE	print contents of a file to console, but only the portion that fits it

Pipe - use the pipe key “|” to transfer output of one command to another.

wc (word count)

wc -l FILE	count the number of lines of a file (-l is “L” not “i”)
cat FILE | **wc** -l	count the number of lines of a file - completely equivalent; **pipe** transfers the output of **cat** to **wc**

sort

cat FILE | **sort**	print the lines of a file sorted by the alphabetical order
cat FILE | **sort -n**	**sort** numbers lowest to highest
cat FILE | **sort -r**	**sort** but reverse order

head / tail

head FILE	print first 10 lines of a file (10 last lines for **tail**)
head -n N FILE	print first N lines of a file (N last lines for **tail**), synonym: **head -**N FILE, where N is a number of lines (same for **tail**)

grep (global regular expression print)

grep “string” FILE	search a FILE for a string and print every *line* it is found in, case sensitive (you can mix the flags), you can use **wildcard** * in place of part of the string
grep -i “string” FILE	**grep** but *ignore case* (StRIng = string)
grep -o “string” FILE	**grep** but print *only* the string each time it is found (not the entire line)
grep -v “string” FILE	**grep** but print every *line that does not* contain the string
grep -r “string” FIL*	**grep** but recursive, search the directory and subdirectories to find any file that matches FIL* and **grep** it. The name of the file will proceed the output of each file (the name can be hidden by **-h** flag)
grep -c “string” FILE	count in how many *lines* the string appears
grep -A N “string” FILE	**grep** but also print N *next lines after* the string
grep -B N “string” FILE	**grep** but also print N *previous lines before* the string
grep -C N “string” FILE	**grep -A** and **grep -B** at the same time, synonym: **grep -**N, where N is the number of lines *before* and *after*
flags: **-E**, **-F**, **-G**, **-P**, **-e**	these flags signify different types of *regular expressions*; this is advanced stuff beyond the scope of this workshop

mc -v (mc viewer)

mc -v FILE	view a file. Move down or up using **arrow keys**. **Page-up / Page-down** to move faster. **Home / End** to go to the top / bottom of a file. More below.

Permissions

chown (change ownership)

chown USER FILE	change FILE’s ownership to USER
chown USER:GROUP FILE	change FILE’s ownership to USER and allow GROUP to access it

chmod (change mode)

chmod u+x FILE	grants you (the user) permission to execute the file
chmod a+rx FILE	grants all people read and execute permission (but not to edit)

Transfer files to or from remote

tar (Tape archiver)

tar -zvcf TAR.GZ FILES*	compress the files matching FILES* to archive TAR.GZ
tar -xvf TAR.GZ	unzip the archive TAR.GZ in current directory

scp (secure copy protocol)

scp -Cpv LOGIN@REMOTE.MACHINE:/path/on/the/remote/machine/FILE* /path/on/your/machine/. - copy files matching FILE* on remote to your machine	
scp -Cpv /path/on/your/machine/FILE* LOGIN@REMOTE.MACHINE:/path/on/the/remote/machine/. - copy files matching FILE* on your machine to the remote	
scp -C	this flag compresses the file during transfer, so its smaller
scp -p	this flag preserves the original modification and creation time of the files
scp -v	this flag gives out more information about the process

Find files

find

find . **-name** FILE	find the FILE in current directory or any sub-directory
find . **-iname** FILE	**find** but case insensitive
find . **-maxdepth** N -iname FILE	**find** but will not enter sub-directories N-lvls deep. 1 means that will not enter any (global flags before -iname flag)
find . **-type** f	find all files recursively
find . **-type** d	find all directories and sub-directories
find . **-mindepth** N **-iname** FILE	like **-maxdepth** but will enter only sub-directories after N-lvls deep

The **find** is a little advanced, so only basics are covered here.

Visual File Manager

mc (midnight commander)

mc -S “dark”	opens **midnight commander** interface
The following keys are usable inside the mc interface. You can still use the console inside mc, but the output will be hidden from you.	
Arrow key up/down	choose a file or directory
Page Up/Down	choose a file or directory *faster*
Home/End	get to the top/bottom of the file list
Enter	on directory: enter that directory, on /..: enter the directory above
TAB	change panel (left or right)
F10	closes the interface
F7	create a directory, like **mkdir**
F9	special options
Following assume, you have a file highlighted.	
F3	view a file
F4	edit a file (mcedit)
F5	copy a file (works on directories too), like **cp**
F6	move a file (works on directories too), like **mv**
F8	remove a file (works on directories too)
Alt+Enter	paste the name of a file to command line
hold Shift + press Arrow key	select group of files (and/or directories)
Other keys	
Alt+H	search previous commands used inside **mc**; similar to Ctrl+R outside
Alt+I	make the panels the same directory (if used on left, then both panels are now in left’s directory)
Alt+O	on directory: open that directory in the other panel on file: open the directory above in the other panel
Enter	on executable file: run the file

Writing files

mc -e (mcedit)

mc -S “dark” -e FILE	opens a FILE to edit using **mcedit**, synonym: **F4** inside **mc**
mc -S “dark” -v FILE	opens a FILE to view, synonym: **F3** inside **mc**
Keys inside **mcedit**	
double ESC	close a file, synonym: **F10**
Ctrl+U	undo the last change, basically like Ctrl+Z on Windows
F2	save the changes
F3, Arrow keys, F3	select text, start/end selection with **F3** select with **Arrow keys,** synonym: **hold Shift + press Arrow key**
F4	search and replace a string
F5	copy selection to where the cursor is currently
F6	move selection to where the cursor is currently
F7	search for a string
F8	delete the selection, **Ctrl+U** will bring it back
F9	special commands
F12	save-file-as
Shift+Insert	paste text from outside **mc** that you have highlighted
Ctrl+Shift+V	paste text from outside **mc** that you copied using **Ctrl+C** outside
Ctrl+F	copy selection to a file, you need to use these keys in sequence; synonyms : **F9, F, Y**
Shift+F5	insert a file where the cursor is, synonym: **F9, F, I** (“i”)
Alt+U	use any outside command inside **mcedit** and print the output where the cursor is
Home/End	go to the start/end of current line
Ctrl+Home/End	go to the start/end of the file
Ctrl+R	redo the last undo, reverse **Ctrl+U**
Alt+L	go to a line by number

Scripting

& - use the “&” at the end of a command to run it in the background.

ps (process status)

ps	shows all running processes, you can see if a background process has ended, gives the PID of running processes

kill

kill PID	kills a process running in the background
kill -9 PID	! use with caution, like **kill**, but you are able to kill a process that couldn’t be canceled using **kill**

bash (bourne again shell, explanation)

./script.sh	run the script.sh (must have a *shebang* at the start and be executable (see **chmod** on page 5), synonym: **Enter** in **mc**
Inside the script	
#!/bin/bash	a *shebang*, put it on the start to make the file a bash script
#	this is used for comments, anything after “#” up to end of a line will not be executed, highlighted in dark orange
$VARIABLE	“$” signifies variables, highlighted in light green
for, in, do etc.	keywords, highlighted in yellow
cd, cp, mkdir etc.	commands, highlighted in blue
“” or ‘’	quotation marks signify strings, highlighted in dark green
``	backquote, grave accent (pl. *grawis*), another way to *subshell*, highlighted in red with black background (black is invisible in mc -S “dark” though)
;	newline marker for scripting, you need a new line before “do” so you can use “;” to make it into one line
Command line scripting, one-line scripting	
code ; *code* ; *code*	you can write a script in the command line, using “;” as a new line. You do not need a *shebang*.

bash (bourne again shell, programming language)

sleep NJ	make the script wait for time N units of time J (default: s)
echo “Hello World”	script will print “Hello World”
date	script will print out current date
VARIABLE=VALUE	to assign a value to a variable you need to use equal sign “=” <u>without spaces before or after</u>
echo $VARIABLE	script will print what is stored in a VARIABLE
DIR=$(**pwd -P**)	so called *subshell*, allows you e.g. to store the output of a command (**pwd -P**) in a variable
DIR=\`**pwd -P**\`	
b=$((1 + a))	The $((...)) construct allows simple arithmetic on **integers**. a is variable, but does not need “$”, because it's within $((...)).

for i in 1 2 3 ; do
echo $i
sleep 5
done
	for loop, the variable $i will be assigned values 1, then 2, then 3 and each time will be printed, each print will be space out by 5 seconds
	for loop must end in “done”

for i in \`**seq** 1 10\` ; do …	**seq** command inside *subshell* will produce a sequence of numbers from 1 to 10 (1 2 3 4 … 10)
for i in \`**seq** 1 2 10\` ; do …	**seq** here will produce a sequence of numbers from 1 to 10, skipping every other number (1 3 5 7 9)
for i in a b abc ; do …	this will go through “a”, then “b”, then “abc”
for i in */ ; do …	this will loop through all directories in current directory; without “/” will go over files too
for i in $(ls -d */); do …	this will loop through all directories in current directory