CSC

ICT Solutions for Brilliant Minds

Working in Unix Command Line

11.3.2019

Working with bash shell

- For more information, see:
  - http://research.csc.fi/csc-guide-linux-basics-for-csc

Unix/linux commands

Basic syntax:

command -option argument

```
ls
ls -l
ls -l myDirectory
```

Use *man* command to get information about possible options

```
man ls
```

Commands for directories:

`cd`	change directory
`ls`	list the contents of a directory
`pwd`	print (=show) working directory
`mkdir`	make directory
`rmdir`	remove directory

Commands for files:

`cat`	print file to screen
`cp`	copy
`less`	view text file
`rm`	remove
`mv`	move/rename a file
`head`	show beginning of a file
`tail`	show end of a file
`grep`	find lines containing given text
`wc`	count number of words or lines
`file`	check the type of the file

Special characters:

*(asterisk), wild card, means any text

```
ls *.fasta
```

| (pipe) guides output of a command to an input of another commands

```
ls *.fasta | less
```

> Writes output to a new file

```
ls > files_of_the_directory.txt
```

~ (tilde) means your home directory as does $HOME

```
cp test.txt ~/file.txt
cp text.txt $HOME
```

& runs command in background

```
gzip my_big_file.tar  &
```

\ (backslash) escape, used to tell the system to ignore special meanings

```
cp this\ filename\ has\ spaces.txt $WRKDIR
```

Piping

- It is also possible to ”pipe” output of one command to another command using ”|” characters
- This can be faster that using files as there is no disk I/O

```
ls –l | less
cat myfile.txt | sort | uniq
```

Redirection

- It is often useful to redirect the output (stdout) of a command to a file
  - o ”>” will overwrite the contents
    - o Try:

```
ls > filelist
cat filelist
```

    - o Depending on your bash settings, may cause error if target file exists
  - o ”>>” will append to a file
- Sometimes it’s necessary to capture stderr as well

```
command > out.file 2> err.file
```

- Both stdout and stderr so same file

```
command &> output.file
command > output.file 2>&1
```

(for older bash versions)

Redirection

- Redirection can also be done in the other direction

  - Redirect the contents of the file to the standard input (stdin) of a command

    ```
    cmd < file
    ```

  - Redirect a bunch of lines to the stdin. If 'EOL' is quoted, text is treated literally.

    ```
    cmd << EOL
    line1
    line2
    EOL
    ```

  - Redirect a single line of text to the stdin of a command

    ```
    cmd <<< "string"
    ```

Variables and arrays

To set a variable:

```
variable=value
```

To use a variable

```
$variable

var1="Hello"
var2="World"
echo $var1 $var2
```

To set an array

```
array=( value1 value2 valueN )
```

To use a value in an array (note: zero based)

```
${array[n]}

array=( a b c )
echo ${array[1]}
```

Variables and arrays

Sometimes it is necessary to separate variable name from rest of the command:

This would not work:

```
sed -n $SLURM_ARRAY_TASK_IDp namelist
```

So instead we can use:

```
sed -n ${SLURM_ARRAY_TASK_ID}p namelist
```

or

```
sed -n ”$SLURM_ARRAY_TASK_ID”p namelist
```

Environment variables

- Normal variables only visible to the process that set them
- To make a variable visible also to any child processes (*e.g.* any programs run from a shell), you must use **`export`** command:

```
export PATH=${PATH}:${USERAPPL}/mcl/version-12-068/bin
```

- Typical examples are the system variables that point to different file system locations: $HOME, $USERAPPL, $WRKDIR etc
- SLURM has it own set of useful system variables: $SLURM_CPUS_PER_TASK, $SLURM_ARRAY_TASK_ID etc

Quotes

- Different quotes have different fuctionalities

”	Take text enclosed within quotes literally
``	Take text enclosed within quotes as command and replace with output
””	Take text within quotes literally after substituting any variables

- Compare the results of these commands:

```
var=”test”; echo ’echo $var’
var=”test”; echo `echo $var`
var=”test”; echo ”echo $var”
```

Some useful commands for parsing lines

Try these to see what they do!

sed

```
echo "one this two this three" | sed s/this/that/
echo "one this two this three" | sed s/this/that/g
```

awk

```
echo ”one two three” | awk ’{print $2}’
echo ”one;two;three” | awk –F”;” ’{print $2 $3}’
```

cut

```
echo ”123456789” | cut –c 4
echo ”123456789” | cut –c -4
echo ”123456789” | cut –c 4-
echo ”123456789” | cut –c 4-7
echo ”one_two_three” | cut –d ”_” –f 2
```

All of these have much more options. See man pages for details.

Some useful commands for parsing lines

grep is a powerful tool for finding regular expressions in files

`grep pattern file`	returns the lines from file containing the pattern
`grep –c pattern file`	returns the count of lines containing the pattern
`grep –v pattern file`	reverses output, *i.e.* returns lines not containing the pattern
`grep –w pattern file`	returns only complete word matches
`grep –f file1 file2`	returns lines in `file2` that also exist in `file1`
`grep ”^pattern” file`	^ matches beginning of line
`grep ”pattern$” file`	$ matches end of line

It’s good to remember that grep operates line by line, *i.e.* matches separated into two lines are not found.

CSC


facebook.com/CSCfi


twitter.com/CSCfi


youtube.com/CSCfi


linkedin.com/company/csc---it-center-for-science


github.com/CSCfi