

Linux Basics III: Text file manipulation

ADRIANO ANGELONE, GRAZIANO GIULIANI

Course Outline

- UNIX/Linux Basics
- Intermediate shell commands
- Editing and compiling source code
- **Text file manipulation**
- Basic shell scripting

Download slides and exercise files with the command

```
git clone https://github.com/AA24KK/LinuxBasics.git
```

or download a ZIP archive at

```
https://github.com/AA24KK/LinuxBasics/archive/master.zip
```

Adriano: **aangelon@ictp.it**, Room 263, ICTP

Graziano: **ggiulian@ictp.it**

Text manipulation

We are scientists: we deal in **datafiles**

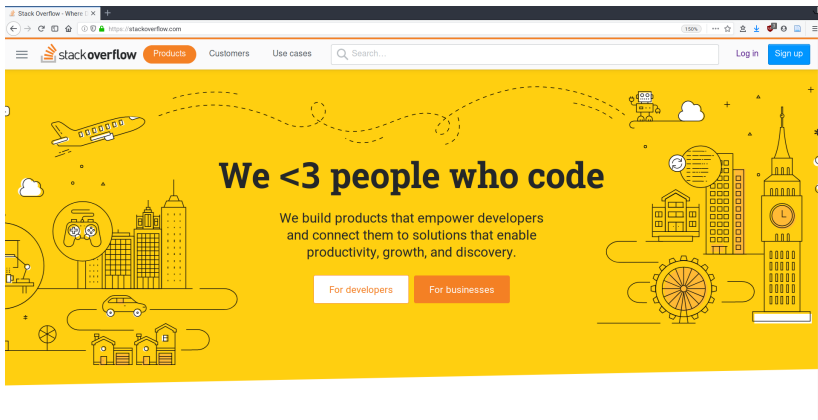
```
mpirun@logint2:/loc-test/na/1123
lock E/N (E/N)^2 M M^2
1 -8.548300e-01 7.307343e-01 9.124911e-01 8.326400e-01
2 -8.552850e-01 7.315124e-01 9.132898e-01 8.340983e-01
3 -8.536700e-01 7.287525e-01 9.112144e-01 8.303117e-01
4 -8.601950e-01 7.399354e-01 9.155645e-01 8.382583e-01
5 -8.521800e-01 7.262108e-01 9.102124e-01 8.284867e-01
6 -8.505400e-01 7.234183e-01 9.098782e-01 8.278783e-01
7 -8.502100e-01 7.228570e-01 9.083235e-01 8.250517e-01
8 -8.497450e-01 7.220666e-01 9.103955e-01 8.288200e-01
9 -8.532500e-01 7.280356e-01 9.112528e-01 8.303817e-01
10 -8.478600e-01 7.188666e-01 9.080189e-01 8.244983e-01
11 -8.557850e-01 7.323680e-01 9.134814e-01 8.344483e-01
12 -8.522500e-01 7.263301e-01 9.111641e-01 8.302200e-01
13 -8.468450e-01 7.171465e-01 9.067800e-01 8.222500e-01
14 -8.472700e-01 7.178665e-01 9.071319e-01 8.228883e-01
15 -8.527000e-01 7.270973e-01 9.116222e-01 8.310550e-01
16 -8.545950e-01 7.303326e-01 9.124537e-01 8.325717e-01
17 -8.516400e-01 7.252907e-01 9.102683e-01 8.285883e-01
18 -8.532950e-01 7.281124e-01 9.124427e-01 8.325517e-01
19 -8.499800e-01 7.224660e-01 9.105429e-01 8.290883e-01
20 -8.537500e-01 7.288891e-01 9.123176e-01 8.323233e-01
21 -8.505350e-01 7.234098e-01 9.108211e-01 8.295950e-01
22 -8.498250e-01 7.222025e-01 9.092048e-01 8.266533e-01
```

Shell commands allow us to manipulate them as **text files**:
great versatility and relatively simple, sometimes requires attention

You see files as a bunch of **rows** or **columns**:
different commands for different tasks

Before we go, remember:
nobody knows everything (except the internet)

Stackoverflow and **Google** will help you, use them



Manual Pages

All basic UNIX commands come with a manual page. The manual page can be accessed through the `man` program.

- `man` is the system manual pager program. You provide as argument the name of a program, utility or function.
- The program searches for the manual page in various section in a pre-defined order.
- The manual page is shown using a pager program after being formatted for the particular terminal output.

Example:

```
man cat
```

Row Operations I - Listing

cat <files> : displays entire files

- **-n** : numbers lines

```
~ > cat example_file
10
20
30
40
50
-----
~ > cat -n example_file
1 10
2 20
3 30
4 40
5 50
```

tail <file> : **last** 10 lines of the file

- **-n <num>** : last **<num>** lines
- **-n +<num>** : after line **<num>**

```
~ > tail -n 3 example_file
30
40
50
-----
~ > tail -n +2 example_file
20
30
40
50
```

head <file> : **first** 10 lines of the file

- **-n <num>** : first **<num>** lines
- **-n -<num>** : before line **<num>**

```
~ > head -n 3 example_file
10
20
30
-----
~ > head -n -3 example_file
10
20
```

Useful on their own, can be combined with **pipes**

Interlude I: Pipes and redirection

Piping:

output of command \longrightarrow input to another



```
command_1 <arguments> | command_2 | ... | command_N
```

Example: extract 3rd line of file

```
head -n 3 <file> | tail -n 1
```

Redirection:

output of command \longrightarrow file

```
command <options> <arguments> > <file>
```

Use `>>` to append to existing file

```
~/test_dir » ls
file_1 file_2 file_3
-----
~/test_dir » ls > log
-----
~/test_dir » ls
file_1 file_2 file_3 log
-----
~/test_dir » cat log
file_1
file_2
file_3
log
```

Interlude II: basic shell scripting

As said before, you can **recycle** commands you use more than once:
write once, use more than once

You can create **scripts**:
files containing instructions
which you launch to perform tasks

```
#!/bin/bash  
echo 'hello'
```

- The first line tells the shell which **interpreter** to use (i.e. which scripting language, we use **bash**)
- The rest is **instructions** (this here prints 'hello')

Scripts can be made **executable**
(remember **chmod**?)
and launched from the command line:
./<script>

```
~ » cat example_script.sh  
#!/bin/bash  
  
echo 'hello'  
-----  
~ » chmod u+x example_script.sh  
-----  
~ » ./example_script.sh  
hello
```


Exercise I - cat, tail, head

Create the following 3 files:

```
~ » cat file_1
1
2
3
4
5
-----
~ » cat file_2
3
4
5
6
7
-----
~ » cat file_3
6
7
8
9
10
```

Write a script that creates a file containing the first 3 rows of **file_1**, the 2nd and 3rd lines of **file_2** and the last 3 lines of **file_3** and displays this file on the screen.

Use pipes and redirection where needed.

Row Operations II - Matching and Filtering

`grep <content> <files>` filters lines based on their content

- `<content>` can be a part of the line
- Quoting (`'<content>'`) is advised
- `-n`: adds numbers to matching lines
- `-i`: case-insensitive matching
- `-v`: prints non-matching lines

```
~ » cat example_file
10 a
20 b
30 I want this one
40 d
50 e
-----
~ » grep 'want' example_file
30 I want this one
-----
~ » grep -n 'want' example_file
3:30 I want this one
-----
~ » grep -n -i 'WANT' example_file
3:30 I want this one
-----
~ » grep -v -i 'WANT' example_file
10 a
20 b
40 d
50 e
```

More flexibility using **regular expressions**

Interlude III - Regular Expressions Basics

Regular expressions (regexps) are **templates** that lines can match
They can use special characters and **wildcards**:

- `.`: any single character
- `.*`: any number of characters
- `^`: beginning of the line
- `$`: end of the line
- `[adf]`, `[a-z]`, `[A-Za-z]`: group of characters

Example: `The quick brown fox jumped`

- `.*quick.*` matches
- `The quick brown.*jumped.*` matches
- `The quick brown [foxape]* jumped .*` matches
- `^quick.*` **doesn't match**

Now you can do `grep <regexp> <files>`:

```
grep '.*quick.*' <files>
```

Exercise II - grep

Create the following file:

```
~/test_dir » cat test
#Index Name Surname Product
1 Robert Duvall Oranges
2 Al Pacino Peaches
#2 Marlon Brando Grapes
2 Diane Keaton Tamarindos
20 Robert DeNiro Cherries
```

Create a script which filters out commented lines (starting with `#`), selects all lines where the index is 2, then selects only who sells tamarindos. Use redirection and/or piping.

Hint: the line begins with the index. Watch case.

Row Operations III - sed

sed (stream editor) operates on files as groups of lines:
finds lines matching regexps and acts on (or around) them

- **sed '/<regexp>/a <text>'**
adds **<text>** after matching lines
- **sed '/<regexp>/i <text>'**
adds **<text>** before matching lines
- **sed '/<regexp>/c <text>'**
replaces matching lines with **<text>**
- **sed '/<regexp>/d'**
deletes all matching lines

```
~ » cat example_file
10
20
30
-----
~ » sed '/2.*/a new' example_file
10
20
new
30
-----
~ » sed '/2.*/i new' example_file
10
new
20
30
-----
~ » sed '/2.*/c new' example_file
10
new
30
-----
~ » sed '/2.*/d' example_file
10
30
```

Row Operations IV - More sed

```
sed 's/<regexp>/<text>/g' <files>
```

replaces **all** occurrence of **<regexp>** with **<text>** in all lines

- Replacement and matching will break words
- Matching is case-sensitive
- All regexp tools available
- **sed -i** applies modifications to the files: **be careful!**

```
~ » cat example_file
is this a test ?
I like apples
the pen is on the table
-----
~ » sed 's/apples/apples and oranges/g' example_file
is this a test ?
I like apples and oranges
the pen is on the table
-----
~ » sed 's/apple/apples and oranges/g' example_file
is this a test ?
I like apples and oranges
the pen is on the table
-----
~ » sed 's/is/IS/g' example_file
IS thIS a test ?
I like apples
the pen IS on the table
-----
~ » sed 's/^is/IS/g' example_file
IS this a test ?
I like apples
the pen is on the table
```

Remember: **sed** can be used in pipes

Exercise III - sed

Create the following file:

```
~ » cat example_file
# Score    Index  Name
0,100      #1     Lucas
0,200      #2     Andrew
#0,400      #3     Mary
0,500      XXX    XXX
0,300      #5     Rose
```

Create a script which:

- Replaces corrupted lines (lines containing `XXX`) with `#CORRUPTED`
- Removes commented lines (beginning with `#`) from the file
- Shows on screen the last two lines of the file replacing `,` with `.`
(**do not apply this last modification to the file**)

Hint: the use of `sed -i` and pipes is suggested.

A copy of the original file is also handy to have at all times.

Column Operations I - cut and paste

Datafiles can also be seen as an ensemble of **columns** (fields)

cut <options> <file>:

extract selected fields from file

- **-d**: specify field delimiter (often **' '** or **','**)
- **-f**: specify the desired fields (separate with **,**)
- **--complement**: print unselected fields

```
~ » cat example_file
1 2 3
10 20 30
100 200 300
-----
~ » cut -d ' ' -f 1,2 example_file
1 2
10 20
100 200
-----
~ » cut -d ' ' -f 1,2 --complement example_file
3
30
300
```

paste <files>:

join lines in multiple files

- **-d**: specify delimiter between files
default: **TAB** (not space!)

```
~ » cat example_file_1
1.0e-1 3.0e-1
2.0e-1 4.0e-1
-----
~ » cat example_file_2
5.0e-1 7.0e-1
6.0e-1 8.0e-1
-----
~ » paste -d ' ' example_file_1 example_file_2
1.0e-1 3.0e-1 5.0e-1 7.0e-1
2.0e-1 4.0e-1 6.0e-1 8.0e-1
```


Column Operations II - sort

sort <options> <file>:

sorts a file according to the given criteria

- **-k**: specify an index column
(order following this column, default: 1)
- **-n**: numbers sorted according to value
- **-g**: like **-n**, more general formats
(e.g., scientific notation)
- **-h**: like **-n**, human-readable formats
(e.g., **4K, 8M**)
- **-r**: reverses sort order (descending)
- **-u**: eliminates repeated lines

```
~ » cat example_file
a 1
C 02
C 02
b 0.5e+00
-----
~ » sort example_file
C 02
C 02
a 1
b 0.5e+00
-----
~ » sort -f example_file
a 1
b 0.5e+00
C 02
C 02
-----
~ » sort -k2 example_file
b 0.5e+00
C 02
C 02
a 1
-----
~ » sort -k2 -g example_file
b 0.5e+00
a 1
C 02
C 02
-----
~ » sort -k2 -g -r example_file
C 02
C 02
a 1
b 0.5e+00
```

Exercise IV - cut, paste, sort

Create the following files:

```
~ » cat example_file_1
1.0e-1 3.0e-1
2.0e-1 4.0e-1
-----
~ » cat example_file_2
5.0e-1 7.0e-1
6.0e-1 8.0e-1
```

Write a script which:

- Pastes the two files together
- Sorts the output according to the 3rd column
- Prints out the 2nd column of the line with the highest value of the 3rd column

Hint: Remember the options of `sort` (`-g` in particular).

Remember `head/tail`.

Column Operations III - awk

awk is a (simple) programming language for text operations
mostly used to work on files as sets of columns

An awk program can be structured in 3 blocks:

```
BEGIN { 1 } { 2 } END { 3 }
```

- **Initial instructions** (1) are executed only once, before starting to read the file.
- **Line instructions** (2) are executed on each line.
- **Final instructions** (3) are executed once the file has been read.

Usually when launched in shell only block (2) is used:

```
awk '{ <commands> }' <file>
```

Powerful tools available, like **if...then...else**
We will not see them here (**stackoverflow** is always there though)

Column Operations IV - awk basics

`print` writes to standard output:

use `""` for strings

Special variables:

- `NR` is the current line
- `NF` is the number of fields of the current line

Access fields via `$<field_number>`

- `$0` is the entire line
- `$NF` is the last field

Fields can be manipulated as strings or floating-point numbers (file remains untouched)

```
~ » cat example_file
a e 1.0
b f 2.0
c g 3.0
d h 4.0

~ » awk '{print NR}' example_file
1
2
3
4

~ » awk '{print NF}' example_file
3
3
3
3
```

```
~ » awk '{print $3}' example_file
1.0
2.0
3.0
4.0

~ » awk '{print $3"-1", $3 - 1.0}' example_file
1.0-1 0
2.0-1 1
3.0-1 2
4.0-1 3
```

Exercise V - awk

Create the following file:

```
~ » cat example_file
# a    b
0.1    1.1
0.2    1.2
0.3    1.3
0.4    1.4
```

Write a script which writes to a new file the row number, the difference and the squared difference of columns 1 and 2 of the starting file (neglecting the label row).

In awk you can perform operations between columns, with the usual operators (+, -, *, /, ()).