# Manipulate text files in bash

The bash language has several commands to read and manipulate txt files, such as: head, tail, more, less, sort, join, wc, uniq. Here we are going to use some of them.

```
cd /media/sf_LVM_shared/my_SE_data/exercise
jupyter-notebook bashinter_osgeo.ipynb
```

## Pattern matching

Create a little file from a large file:

[4]:
```
! head -1000 txt/aver_month_nuts3_fire.asc > input.txt
```

Read/explore the input.txt file

[5]:
```
! head input.txt
```

```
NUTS YYYY MM 0 BAREA
BG311 2005 04 2 0.282594
BG311 2006 11 2 0.600812
BG311 2007 01 3 65.8331
BG311 2007 02 3 9.78246
BG311 2007 04 2 44.4997
BG311 2007 06 2 30.5861
BG311 2007 07 2 5534.21
BG312 2005 04 3 10.6419
BG312 2006 10 2 0.293182
```

[6]:
```
! tail input.txt
```

```
DE425 2000 06 4 3.1973
DE425 2000 07 4 0.724873
DE425 2000 08 4 4.67528
DE425 2000 09 3 0.194243
DE425 2001 04 2 0.0724194
DE425 2001 05 2 0.66708
DE425 2001 08 2 0.0421668
DE425 2002 02 2 0.0125149
DE425 2002 03 2 0.492932
DE425 2002 04 4 1.06466
```

Count the line/word/character in a input.txt

[7]:
```
! wc input.txt
```

```
 1000  5000 24535 input.txt
```

Search for a word in a file

[9]:
```
%%bash
grep "2007" input.txt | head
```

```
BG311 2007 01 3 65.8331
BG311 2007 02 3 9.78246
BG311 2007 04 2 44.4997
BG311 2007 06 2 30.5861
BG311 2007 07 2 5534.21
BG312 2007 01 3 114.535
BG312 2007 02 3 17.3247
BG312 2007 03 3 322.063
BG312 2007 04 3 521.189
BG312 2007 05 3 4.13178
```

# Sorting a file

I want to search for a command able to sort the input.txt table based on the Year column (YYYY).

[10]:
```
! man -k  sort
```

```
apt-sortpkgs (1)     - Utility to sort package index files
bunzip2 (1)          - a block-sorting file compressor, v1.0.6
bzip2 (1)            - a block-sorting file compressor, v1.0.6
comm (1)             - compare two sorted files line by line
heapsort (3bsd)      - sort functions
mergesort (3bsd)     - sort functions
osmium-merge (1)     - merge several sorted OSM files into one
osmium-sort (1)      - sort OSM files
otbcli_ConvertSensorToGeoPoint (1) - OTB ConvertSensorToGeoPoint application
otbgui_ConvertSensorToGeoPoint (1) - OTB ConvertSensorToGeoPoint application
radixsort (3bsd)     - radix sort
sort (1)             - sort lines of text files
sortshp (1)          - sort a Shape data set
sradixsort (3bsd)    - radix sort
tsort (1)            - perform topological sort
```

One of the last lines contain: sort (1) - sort lines of text files So i will search how to use the sort command:

[11]:
```
! man sort
```

NAME
sort - sort lines of text files

SYNOPSIS
sort [OPTION]... [FILE]...
sort [OPTION]... --files0-from=F

DESCRIPTION
Write sorted concatenation of all FILE(s) to standard output.

With no FILE, or when FILE is -, read standard input.

Mandatory arguments to long options are mandatory for short options too. Ordering options:

-b, --ignore-leading-blanks
ignore leading blanks

-d, --dictionary-order
consider only blanks and alphanumeric characters

-f, --ignore-case
fold lower case to upper case characters

-g, --general-numeric-sort
compare according to general numerical value

-i, --ignore-nonprinting
consider only printable characters

-M, --month-sort
compare (unknown) < 'JAN' < ... < 'DEC'

-h, --human-numeric-sort
compare human readable numbers (e.g., 2K 1G)

-n, --numeric-sort
compare according to string numerical value

-R, --random-sort
shuffle, but group identical keys. See shuf(1)

--random-source=FILE
get random bytes from FILE

-r, --reverse
reverse the result of comparisons

--sort=WORD
sort according to WORD: general-numeric -g, human-numeric -h, month -M, numeric -n, random -R, version -V

-V, --version-sort
natural sort of (version) numbers within text

Other options:

--batch-size=NMERGE
merge at most NMERGE inputs at once; for more use temp files

-c, --check, --check=diagnose-first
check for sorted input; do not sort

-C, --check=quiet, --check=silent
like -c, but do not report first bad line

--compress-program=PROG
compress temporaries with PROG; decompress them with PROG -d

--debug

annotate the part of the line used to sort, and warn about questionable usage to stderr

--files0-from=F
read input from the files specified by NUL-terminated names in file F; If F is - then read names from standard input

-k, --key=KEYDEF
sort via a key; KEYDEF gives location and type

-m, --merge
merge already sorted files; do not sort

-o, --output=FILE
write result to FILE instead of standard output

-s, --stable
stabilize sort by disabling last-resort comparison

-S, --buffer-size=SIZE
use SIZE for main memory buffer

-t, --field-separator=SEP
use SEP instead of non-blank to blank transition

-T, --temporary-directory=DIR
use DIR for temporaries, not $TMPDIR or /tmp; multiple options specify multiple directories

--parallel=N
change the number of sorts run concurrently to N

-u, --unique
with -c, check for strict ordering; without -c, output only the first of an equal run

-z, --zero-terminated
line delimiter is NUL, not newline

--help display this help and exit

--version
output version information and exit

KEYDEF is F[.C][OPTS][,F[.C][OPTS]] for start and stop position, where F is a field number and C a character position in the field; both are origin 1, and the stop position defaults to the line's end. If neither -t nor -b is in effect, characters in a field are counted from the beginning of the preceding whitespace. OPTS is one or more single-letter ordering options [bdfgiMhnRrV], which override global ordering options for that key. If no key is given, use the entire line as the key. Use --debug to diagnose incorrect key usage.

SIZE may be followed by the following multiplicative suffixes: % 1% of memory, b 1, K 1024 (default), and so on for M, G, T, P, E, Z, Y.

*** WARNING *** The locale specified by the environment affects sort order. Set LC_ALL=C to get the traditional sort order that uses native byte values.

AUTHOR
Written by Mike Haertel and Paul Eggert.

REPORTING BUGS
GNU coreutils online help: <http://www.gnu.org/software/coreutils/>
Report sort translation bugs to <http://translationproject.org/team/>

COPYRIGHT

```
SEE ALSO
       shuf(1), uniq(1)

       Full documentation at: <http://www.gnu.org/software/coreutils/sort>
       or available locally via: info '(coreutils) sort invocation'

GNU coreutils 8.28               January 2018                            SORT(1)
```

The -k option identify the column of sorting:
Sorting based on column number 2 ( -k 2,2)
sorting based on column number 2 and then number 1 ( -k 2,1)
See again man sort for more options like -n -g
Alfa numeric sorting:

[12]:
```
! sort -k 2,2 input.txt | head
```

```
DE121 1997 05 2 0.232016
DE122 1997 05 2 0.0637817
DE124 1997 03 2 1.28501
DE125 1997 05 2 0.107349
DE128 1997 04 2 0.340913
DE129 1997 03 2 0.297982
DE12A 1997 03 2 0.0815152
DE123 1998 04 2 0.434829
DE124 1998 03 2 0.0796515
DE12A 1998 05 2 0.345525
sort: write failed: 'standard output': Broken pipe
sort: write error
```

General numerical sorting

[13]:
```
! sort -k 2,2 -g  input.txt | head
```

```
NUTS YYYY MM 0 BAREA
DE121 1997 05 2 0.232016
DE122 1997 05 2 0.0637817
DE124 1997 03 2 1.28501
DE125 1997 05 2 0.107349
DE128 1997 04 2 0.340913
DE129 1997 03 2 0.297982
DE12A 1997 03 2 0.0815152
DE123 1998 04 2 0.434829
DE124 1998 03 2 0.0796515
sort: write failed: 'standard output': Broken pipe
sort: write error
```

String numerical sorting

[14]:
```
! sort -k 2,2 -n  input.txt | head
```

```
NUTS YYYY MM 0 BAREA
DE121 1997 05 2 0.232016
DE122 1997 05 2 0.0637817
DE124 1997 03 2 1.28501
DE125 1997 05 2 0.107349
DE128 1997 04 2 0.340913
DE129 1997 03 2 0.297982
DE12A 1997 03 2 0.0815152
DE123 1998 04 2 0.434829
DE124 1998 03 2 0.0796515
sort: write failed: 'standard output': Broken pipe
sort: write error
```

Save the result of a command in a file by ““>”” symbol

[15]:
```
! sort -k 2,2 -g input.txt > input_s.txt
! wc -l input_s.txt
```

```
1000 input_s.txt
```

Which is the first and last year of observations?

# Append the command result to a file

Add the result of a command in the already existing “output” file by ‘>>’ symbol

[16]:
```
! sort -k 3,3 -g input.txt >> input_s.txt
! wc -l input_s.txt
```

```
2000 input_s.txt
```

# Use the variable

Define the value of the variable, print it by putting it in front of the $ symbol

[18]:
```
! var=21
! echo $var
```

Define the value of the variable using the result of a command

[19]:
```
%%bash
var=$(grep "2007" input.txt | wc -l)
echo $var
```

```
189
```

# For loop

In computer science, a for-loop (or simply for loop) is a control flow statement for specifying iteration, which allows code to be executed repeatedly (source https://en.wikipedia.org/wiki/For_loop). We want to automatically count how many observations exist in the years 2007, 2006 and 2005 in the input.txt file. To solve this task we can use the variable and list word/number loop function

[21]:
```
%%bash
for var in 2005 2006 2007; do
    grep $var input.txt
done | head
```

```
BG311 2005 04 2 0.282594
BG312 2005 04 3 10.6419
BG313 2005 03 2 48.0927
BG314 2005 03 4 2.21985
BG315 2005 03 2 125.772
BG315 2005 04 2 95.5232
BG315 2005 06 2 3.70607
BG321 2005 03 3 59.201
BG321 2005 04 2 0.562725
BG322 2005 03 3 6.33855
```

and now we count

[22]:
```
%%bash
for var in 2005 2006 2007; do
    grep $var input.txt | wc -l
done
```

```
121
280
189
```

Now we want to automatically count and save in a file how many observations exist from year 2000 to 2008 in input.txt file. For this use the serial number list loop function.

[24]:
```
%%bash
rm -f input_wc.txt
for ((var=2000 ; var<=2008 ; var++)); do
    grep $var input.txt | wc -l  >> input_wc.txt
done
```

[25]:
```
! head input_wc.txt
```

```
62
34
48
93
46
121
280
189
2
```

# If condition in a for loop

As for the previews exercise, we want to automatically count how many observations exist from year 2000 to 2008 in input.txt file, but not for the year 2003. For this you should use the serial number list loop function with the if condition.

[26]:
```
%%bash
rm no2003output.txt
for ((year=2000 ; year<=2008 ; year++)); do
if [ $year != 2003 ] ; then
    grep " $year " txt/aver_month_nuts3_fire.asc  | wc -l  >> no2003output.txt
fi
done
cat no2003output.txt
```

```
2778
2643
2641
2894
2837
3011
775
0
rm: cannot remove 'no2003output.txt': No such file or directory
```

The same operation can be done by saving the file outised the for loop. When to use the first approch and when to use the second one?

[29]:
```
%%bash
rm no2003output.txt
for ((year=2000 ; year<=2008 ; year++)); do
if [ $year != 2003 ] ; then
    grep " $year " txt/aver_month_nuts3_fire.asc  | wc -l
fi
done > no2003output.txt
cat no2003output.txt
```

```
2778
2643
2641
2894
2837
3011
775
0
```

I need to know in each year which was the biggest fire and print it. I can use the sort command and get the largest fire in the last position.

[30]:
```
%%bash
for ((year=2000 ; year<=2008 ; year++)); do
if [ $year != 2003 ] ; then
    grep " $year " txt/aver_month_nuts3_fire.asc  | sort -k 5,5 -g | tail -1
fi
done
```

```
GR253 2000 07 3 23216.3
PT117 2001 08 464 7226.27
PT118 2002 08 448 14574.7
PT150 2004 07 5 16599
PT164 2005 08 114 41830.3
ES114 2006 08 554 52093.4
BG422 2007 08 4 12972.6
```

### Exercise

Perform the same loop but excluding the year from 2002 to 2004. Use the “man test” to see the option for the if condition. Googled “if statement with multiple condition bash”.

## Checking the flow statement

How can I check that the results are correct and that i’m using the correct variables? By printing the variable during the process and if you need also in the file.

[31]:
```
%%bash
rm no2003output.txt
for ((year=2000 ; year<=2008 ; year++)); do
if [ $year != 2003 ] ; then
    echo processing year $year
    grep " $year " txt/aver_month_nuts3_fire.asc  | wc -l >> no2003output.txt
fi
done
```

```
processing year 2000
processing year 2001
processing year 2002
processing year 2004
processing year 2005
processing year 2006
processing year 2007
processing year 2008
```

[33]:
```
! head no2003output.txt
```

```
2778
2643
2641
2894
2837
3011
775
0
```

I can also run manually a command and compare the results.

[34]:
```
! grep " 2007 " txt/aver_month_nuts3_fire.asc  | wc -l
! grep " 2002 " txt/aver_month_nuts3_fire.asc  | wc -l
```

```
775
2641
```

[35]:
```
%%bash
time for ((year=2000 ; year<=2008 ; year++)); do
if [ $year != 2003 ] ; then
    echo year $year
    grep " $year " txt/aver_month_nuts3_fire.asc  | wc
fi
done
```

```
year 2000
   2778   13890   67910
year 2001
   2643   13215   64585
year 2002
   2641   13205   64534
year 2004
   2894   14470   70924
year 2005
   2837   14185   69493
year 2006
   3011   15055   73972
year 2007
    775    3875   19026
year 2008
      0       0       0


real    0m0.132s
user    0m0.021s
sys     0m0.036s
```

# Debugging

The shell reports message and status symbols in case of error syntax, incorrect commands or inexistent files. Here are reported the most common errors using the example:

[36]:
```
%%bash
for ((year=2000 ; year<=2008 ; year++)); do
    grep " $year " txt/aver_month_nuts3_fire.asc  | wc -l
done
```

```
2778
2643
2641
3078
2894
2837
3011
775
0
```

Run the script and see the error results.

The loop was not close and after the bash error a series of no sense python errors are reported.

[37]:
```
%%bash
for ((year=2000 ; year<=2008 ; year++)); do
    grep " $year " txt/aver_month_nuts3_fire.asc  | wc -l
```

```
bash: line 3: syntax error: unexpected end of file

---------------------------------------------------------------------------
CalledProcessError                        Traceback (most recent call last)
<ipython-input-37-94fe56a6d559> in <module>
----> 1 get_ipython().run_cell_magic('bash', '', 'for ((year=2000 ; year<=2008 ;
year++)); do\n    grep " $year " txt/aver_month_nuts3_fire.asc  | wc -l\n')

~/miniconda3/lib/python3.8/site-packages/IPython/core/interactiveshell.py in
run_cell_magic(self, magic_name, line, cell)
   2397             with self.builtin_trap:
   2398                 args = (magic_arg_s, cell)
-> 2399                 result = fn(*args, **kwargs)
   2400             return result
   2401

~/miniconda3/lib/python3.8/site-packages/IPython/core/magics/script.py in
named_script_magic(line, cell)
    140             else:
    141                 line = script
--> 142             return self.shebang(line, cell)
    143
    144         # write a basic docstring:

~/miniconda3/lib/python3.8/site-packages/decorator.py in fun(*args, **kw)
    230             if not kwsyntax:
    231                 args, kw = fix(args, kw, sig)
--> 232             return caller(func, *(extras + args), **kw)
    233     fun.__name__ = func.__name__
    234     fun.__signature__ = sig

~/miniconda3/lib/python3.8/site-packages/IPython/core/magic.py in <lambda>(f, *a, **k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188
    189         if callable(arg):

~/miniconda3/lib/python3.8/site-packages/IPython/core/magics/script.py in shebang(self,
line, cell)
    243             sys.stderr.flush()
    244         if args.raise_error and p.returncode!=0:
--> 245             raise CalledProcessError(p.returncode, cell, output=out,
stderr=err)
    246
    247     def _run_script(self, p, cell, to_close):

CalledProcessError: Command 'b'for ((year=2000 ; year<=2008 ; year++)); do\n    grep "
$year " txt/aver_month_nuts3_fire.asc  | wc -l\n'' returned non-zero exit status 2.
```

Bash: syntax error near unexpected token `('

The error is near the brackets. Often it is just a space or a bracket that has not been closed

[38]:
```
%%bash
for ( (year=2000 ; year<=2008 ; year++)); do
    grep " $year " txt/aver_month_nuts3_fire.asc  | wc -l
done
```

```
bash: line 1: syntax error near unexpected token `('
bash: line 1: `for ( (year=2000 ; year<=2008 ; year++)); do'
```

```
---------------------------------------------------------------------------
CalledProcessError                        Traceback (most recent call last)
<ipython-input-38-186e4e61dddc> in <module>
----> 1 get_ipython().run_cell_magic('bash', '', 'for ( (year=2000 ; year<=2008 ;
year++)); do\n    grep " $year " txt/aver_month_nuts3_fire.asc  | wc -l  \ndone   \n')

~/miniconda3/lib/python3.8/site-packages/IPython/core/interactiveshell.py in
run_cell_magic(self, magic_name, line, cell)
   2397             with self.builtin_trap:
   2398                 args = (magic_arg_s, cell)
-> 2399                 result = fn(*args, **kwargs)
   2400             return result
   2401

~/miniconda3/lib/python3.8/site-packages/IPython/core/magics/script.py in
named_script_magic(line, cell)
    140             else:
    141                 line = script
--> 142             return self.shebang(line, cell)
    143
    144         # write a basic docstring:

~/miniconda3/lib/python3.8/site-packages/decorator.py in fun(*args, **kw)
    230             if not kwsyntax:
    231                 args, kw = fix(args, kw, sig)
--> 232             return caller(func, *(extras + args), **kw)
    233     fun.__name__ = func.__name__
    234     fun.__signature__ = sig

~/miniconda3/lib/python3.8/site-packages/IPython/core/magic.py in <lambda>(f, *a, **k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188
    189         if callable(arg):

~/miniconda3/lib/python3.8/site-packages/IPython/core/magics/script.py in shebang(self,
line, cell)
    243             sys.stderr.flush()
    244         if args.raise_error and p.returncode!=0:
--> 245             raise CalledProcessError(p.returncode, cell, output=out,
stderr=err)
    246
    247     def _run_script(self, p, cell, to_close):

CalledProcessError: Command 'b'for ( (year=2000 ; year<=2008 ; year++)); do\n    grep "
$year " txt/aver_month_nuts3_fire.asc  | wc -l  \ndone   \n'' returned non-zero exit
status 2.
```

Bash command error: use “man -k” for searching the operation that you need.

[39]:
```
%%bash
for ((year=2000 ; year<=2008 ; year++)); do
    grap " $year " txt/aver_month_nuts3_fire.asc  | wc -l
done
```

```
0
0
0
0
0
0
0
0
0
```

```
bash: line 2: grap: command not found
bash: line 2: grap: command not found
bash: line 2: grap: command not found
bash: line 2: grap: command not found
bash: line 2: grap: command not found
bash: line 2: grap: command not found
bash: line 2: grap: command not found
bash: line 2: grap: command not found
bash: line 2: grap: command not found
```

Invalid command option: “wc: invalid option – ‘k’”. Read carefully the manual for the wc command.

[40]:
```
import warnings; warnings.simplefilter('ignore')
```

[41]:
```
%%bash
for ((year=2000 ; year<=2008 ; year++)); do
    grep " $year " txt/aver_month_nuts3_fire.asc  | wc -k
done
```

```
wc: invalid option -- 'k'
Try 'wc --help' for more information.
wc: invalid option -- 'k'
Try 'wc --help' for more information.
wc: invalid option -- 'k'
Try 'wc --help' for more information.
wc: invalid option -- 'k'
Try 'wc --help' for more information.
wc: invalid option -- 'k'
Try 'wc --help' for more information.
wc: invalid option -- 'k'
Try 'wc --help' for more information.
wc: invalid option -- 'k'
Try 'wc --help' for more information.
wc: invalid option -- 'k'
Try 'wc --help' for more information.
wc: invalid option -- 'k'
Try 'wc --help' for more information.
```

```
---------------------------------------------------------------------------
CalledProcessError                        Traceback (most recent call last)
<ipython-input-41-74385b15380a> in <module>
----> 1 get_ipython().run_cell_magic('bash', '', 'for ((year=2000 ; year<=2008 ;
year++)); do\n    grep " $year " txt/aver_month_nuts3_fire.asc  | wc -k\ndone\n')

~/miniconda3/lib/python3.8/site-packages/IPython/core/interactiveshell.py in
run_cell_magic(self, magic_name, line, cell)
   2397             with self.builtin_trap:
   2398                 args = (magic_arg_s, cell)
-> 2399                 result = fn(*args, **kwargs)
   2400             return result
   2401

~/miniconda3/lib/python3.8/site-packages/IPython/core/magics/script.py in
named_script_magic(line, cell)
    140             else:
    141                 line = script
--> 142             return self.shebang(line, cell)
    143
    144         # write a basic docstring:

~/miniconda3/lib/python3.8/site-packages/decorator.py in fun(*args, **kw)
    230             if not kwsyntax:
    231                 args, kw = fix(args, kw, sig)
--> 232             return caller(func, *(extras + args), **kw)
    233     fun.__name__ = func.__name__
    234     fun.__signature__ = sig

~/miniconda3/lib/python3.8/site-packages/IPython/core/magic.py in <lambda>(f, *a, **k)
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188
    189         if callable(arg):

~/miniconda3/lib/python3.8/site-packages/IPython/core/magics/script.py in shebang(self,
line, cell)
    243             sys.stderr.flush()
    244         if args.raise_error and p.returncode!=0:
--> 245             raise CalledProcessError(p.returncode, cell, output=out,
stderr=err)
    246
    247     def _run_script(self, p, cell, to_close):

CalledProcessError: Command 'b'for ((year=2000 ; year<=2008 ; year++)); do\n    grep "
$year " txt/aver_month_nuts3_fire.asc  | wc -k\ndone\n'' returned non-zero exit status
1.
```

The file or directory does not exist: search for the correct file and directory, by using “cd” and “pwd”

[42]:
```
%%bash
for ((year=2000 ; year<=2008 ; year++)); do
    grep " $year " ../aver_month_nuts3_fire.asc  | wc -l
done
```

```
0
0
0
0
0
0
0
0
0
```

```
grep: ../aver_month_nuts3_fire.asc: No such file or directory
grep: ../aver_month_nuts3_fire.asc: No such file or directory
grep: ../aver_month_nuts3_fire.asc: No such file or directory
grep: ../aver_month_nuts3_fire.asc: No such file or directory
grep: ../aver_month_nuts3_fire.asc: No such file or directory
grep: ../aver_month_nuts3_fire.asc: No such file or directory
grep: ../aver_month_nuts3_fire.asc: No such file or directory
grep: ../aver_month_nuts3_fire.asc: No such file or directory
grep: ../aver_month_nuts3_fire.asc: No such file or directory
grep: ../aver_month_nuts3_fire.asc: No such file or directory
```

## remove processed files

[43]:
```
! rm input_s.txt  input.txt input_wc.txt no2003output.txt
```