REGEXES

letters, numbers whitespaces = characters

The best practice is to write as specific regular expressions as possible, to not get false positives.

\d	• Any digit from 0 to 9 • Symbol „\“ used to distinguish the command from simple letter „**d**“
.	• Dot character is a wildcard symbol • It can match any single character (letter, digit, whitespace etc. - anything) • In order to specifically match a dot in some string you should use “**\\.**”

[…]

- Squared brackets are used as method of matching specific characters

Use:

[abc]	This patter will only a single character, “a” or “b” or “c”

Example:

Strings	Suitable regex	Match/no-match
can	[cnm][a][n]	yes
man	or	yes
fan	[cnm]an	yes

- Note that $\boxed{\text{[ab] !=[a][b]}}$, as the first regex will match a single string which is either “a” or “b”, while the second regex will match two character of which first has to be “a” and the second has to be “b”.

[^…]

- alt + 94 = “^” sign
- Combination of cap sign “^” inside squared brackets is used for excluding specific characters

Use:

[^abc]	This patter will match any single character, except “a” and “b” and “c”

Example:

Strings	Suitable regex	Match/no-match
log	[^r]og	yes
for		yes
rog		no

-

- Dash character is used inside brackets between two character, to specify that we are looking for a sequence (or range) of characters.
- Multiple character ranges can be used in the same brackets
- Character ranges inside brackets can be used in combination with single characters

Use:

[0-6]	This patter will match any single character which is in range from “0” to “6”
[^n-p]	This patter will match any single character, except characters which is in range from “n” to “p” (these characters are: “n”, “o” and “p”)
[A-Za-z0-9]	This patter will match any single character which is in ranges: 1. from “A” to “Z” 2. from “a” to “z” 3. from “0” to “9”

Example:

Strings	Suitable regex	Match/no-match
Ana	[A-Z]	yes
Bob		yes
Cpc		yes
aax		no

- Note that $\boxed{\text{[A-Za-z] != [A-Z][a-z]}}$, as the first regular expression matches one character, which belongs either to a range between “A” and “Z” or range between “a” and “z”, while on the other hand the second regular expression matches two character, where the first is belongs to range between “A” and “Z” which is followed by a character which belongs to range between “a” and “z”

character{number}

- Specifies how much repetitions of a certain character we want recognise in a string

Example:

Strings	Suitable regex	Match/no-match
aaa	a{3}	yes
aab		no
abb		no

character{number, number}

- Specifies a range of repetitions of a certain character

Example:

Strings	Suitable regex	Match/no-match
aaa	a{1,3}	yes
aab		yes
abb		yes

Example 2:

Strings	Suitable regex	Match/no-match
wxy	[wxy]{5}	no
wwwww		yes
wxywxy		yes

Example 3:

Strings	Suitable regex	Match/no-match
wazzzzzzzap	waz{2,6}up	yes
wazzap		yes
wazap		no

*

- Star character is used to describe that there can be zero or more repetitions of some character or sequence of characters

Use:

\d*	This patter will match zero or more consecutive repetitions of any number

+

- Plus character is used to describe that there can be one (not zero) or more repetitions of some character or sequence of characters

Use:

\d+	This patter will match one or more consecutive repetitions of any number

Example:

Strings	Suitable regex	Match/no-match
aaaabc	aa+b*c+	yes
aabbbbc		yes
aacc		yes
a		no

?	• Question mark character is used when we want to allow one or none occurrences of a character or sequence of characters Example:

Strings	Suitable regex	Match/no-match
abc	ab?c	yes
ac		yes

	• In order to match a question mark character ”**?**” in a string, you should use “**\?**” Example:

Strings	Suitable regex	Match/no-match
1 file found?	[1-9]+ files? found\?	yes
24 files found?		yes
no files found		no

\s	• This will match any kind of whitespace (space, tap, new line…)
\t	• Will match tab whitespace
\n	• Will match new line whitespace
“ ”	• The space whitespace character • It is written inside of quotation marks so it could be seen, but when writing a regex, you don’t write the quotation marks Example:

Strings		Suitable regex	Match/no-match
1. abc	there is a space after “1.”	\d+\.\s+abc	yes
2. abc	there is a tab after “2.”		yes
3. abc	there is a new line after “3.”		yes
4.abc	there is no whitespace after “4.”		no

^	• The cap character, when not used inside of squared brackets, is used to describe that the regex that we are looking for is referring to the beginning of the string

$

- The dollar sign character is used to describe that the regex that we are looking for referring to the end of the string

Use:

^success	This patter will match only strings which begin with word “success”

Example:

Strings	Suitable regex	Match/no-match
Mission: successful	^Mission: successful$	yes
Last mission: unsuccessful		no
Next mission: successful upon catching a target		no

Note: if we wrote the following regex ^Mission ,we would get the same output for the given strings. On the other hand, if we used the following regex successful$, we would not get the same matchings as in the example above, because then for the first two strings the value in “match/no-match” column would be “yes”

(…)

- Expression in the round brackets is extracted for further processing (so you can for an example return the extracted values or values as a return value from a function)

Use:

^(IMG.*\.png)$	Extract all images names, including the extension
^(IMG.*)\.png$	Extract all images names, but not including the extension

Example:

Strings	Suitable regex	Match/no-match	Extracted
file_123456.pdf	^(file.*)\.pdf$	yes	file_123456
file_something.pdf		yes	file_something
testfile.pdf.tmp		no	

(…(…)…)

- When writing single round brackets, inside a part of a regex which is already inside round brackets leads to extracting multiple layers information, which results in nested groups

Use:

^(IMG(\d+))\.png	This patter will extract both the full name of an image, and as well it will extract the picture number from its name If it was used on a string from a previous example with just a different extension, “png” instead of “pdf”, so the string would be ”file_123456”, then the extracted strings would be “file_123456” and “123456”

Example:

Strings	Suitable regex	Match/no-match	Extracted	
Jan 1987	^([A-Z][a-z]{2})(\d+))$	yes	Jan 1987	1987
May 1969		yes	May 1969	1969
Aug 2011		yes	Aug 2011	2011

(…|…)

- When the pipe character is used inside round brackets, then it has a a meaning of logical OR, allowing different character or sets of characters to match the regular expression

Example:

Strings	Suitable regex	Match/no-match
Buy more milk	Buy more (bread|juice|milk)	yes
Buy more bread		yes
Buy more juice		yes

Example 2:

Strings	Suitable regex	Match/no-match
cats	([cb]ats*|[dh]ogs?)	yes
dogs		yes
hogs		yes
bats		yes

Example 3:

Strings	Suitable regex	Match/no-match
I love cats	I love (cats|dogs)	yes
I love dogs		yes
I love hogs		no
I love bats		no

Example:

The task is to extract the value from the 6th column:

Strings	Suitable regex	Match/no-match
1|1234|5693|7459|8246|531|792489|abc|cba (the above string is in one row, but there was not enough space to put it in one row)	^[^|]*{5}\\([0-9]*\\).*	Error: Invalid target and quantifier, because "{number}" can't go after "*"
	^([^|]*){5}\\([0-9]*\\).*	Round brackets when used in regular expressions don't have the same meaning as in math, meaning that they are not used to group part of a regex, so we can't use "{number}" after it.
	^[^|]*[^|]*[^|]*[^|]*[^|]*\\([0-9]*\\).*	Not working properly as pipe symbols "|" are not read
	^[^|]*|[^|]*|[^|]*|[^|]*|[^|]*|\\([0-9]*\\).*	working

BASH COMMANDS

In this file a shade of grey color will be used to denote that something should be written instead of the text provided in the grey color (example of grey text: „command“).

Shell script is:

1. A plain text file which contains a series of commands
2. Anything that you can run in a the Terminal (which is Linux equivalent to Windows command line) can be put into a script, and it will do exactly the same thing
3. Linux is an extensionless operating system, so it is not necessary to put an extension on a bash file, but convention is that you put the ".sh" extension
4. Before executing a bash script, it must have a permission to be executed, which is not given to a bash script by default because of security reasons
5. Command for granting permission to a bash script to be executed can be done by using the command: "chmod 755 script_name", where instead of script_name we use the real name of the script we created
6. Command for executing a bash script is: "./script_name", where we again instead of script_name type in the real name of the script we created.

When after a command we write a dash symbol "-", this means that we are using some specific feature of this command, and that the command should be interpreted differently than as if we wrote it without this dashed symbol. Dashed symbol if always followed with a letter (without putting space in between them), and these two characters combined are called **flags**. An important note is that almost every command has flags and that in one line you can more than one flag of a command, meaning that one flag most often does not exclude the other flags of a command.

	Expression	Meaning
1	ls	Linux shell command which lists directory contents, i.e. files and directories which are visible from the current directory you are positioned in.
2	cd directory OR cd “directory”	These two commands have the same purpose, and that is to position yourself inside a wanted directory. Both commands can be used not just to enter a directory which is just one level above the directory you are currently positioned in. The only difference between these two commands is that the version of the command with the quotation marks can be used when there are whitespaces inside the directory path we want to position ourselves in.
3	clear	Clears the terminal from all that is outputted in it.
4	reset	Resets the configuration of your terminal.
5	man command	Name of the command “man” is short for “manual”, and it gives a short description of what does command do, and what are the flags which can go along with the command.
6	echo ‘string’	Using the command whatever string you put instead of string will be outputted in the terminal.
7	<(command)	This expression tells the command interpreter to run command and make its output appear as a file. The command can be any arbitrarily complex shell command.
8	command1 | command2	When using a pipe symbol, the result of command1 is going to be the input into command2.
9	command > file	The result of command is written into file. An important thing to mention here is that the output will overwrite anything that is written inside the file.
10	command >> file	The result of command is written into file, but in such a manner that the result will be appended to the content which is already in the file.
11	command < file	Run the command with the input coming from the file.
12	command 2> /dev/null	„2> /dev/null“ means to redirect standard error to „/dev/null“. „/dev/null“ is a special device that discards everything that is written to it.
13	cat file	Read the content of the file.
14	cat file1 file2 file3	Concatenating multiple files together.
15	command | cat	Reading the content of the output of the command.

16	cat -v binaryfile	Escaping non-printable binary data. Reading binary files with command “cat” can be risky because it can mess connection to keyboard, as the keyboard is going to try to interpret the content which has symbols not present on the keyboard. If you want to make sure that your Terminal is showing the correct content after using “cat” command on a binary file, use command “reset”, and command “clear” afterwards.
17.1	zcat	These three commands are used for decompressing the file for the result, while the files stay compressed on the disk. The names of compressed input files are expected to end in „.Z“, „.gz“, or „.bz2“. If a specified input file name does not end in this suffix, „zcat“ searches for a file named „*file*.Z“, „*file*.gz“, or „*file*.bz2“. For example, if the command line specifies file abc, zcat looks for „abc.Z“, „abc.gz“, or „abc.bz2“. If none of these are found, zcat uses abc with no extension. The “xzcat” command is used when you want to unzip a file with extension “.xz”.
17.2	bzcat	
17.3	xzcat	

18	more file	Using command “more” we read content of a file, but we read the entire file at once and then navigate through it.
19	less file	Name of the command “less” was given as a joke because command “more” was used for the same purpose except for the fact that if you use “less”, it is faster. The reason for this is that it does not load the entire file at once, and this as well allows navigation through the file using the keys up and down.
20	command | less	Command less can also be used in combination with the pipe symbol, which would mean that it can accept as an input the output of the command

21	grep regex file	Name of command “grep” stands for: “**global regular expression print**”. Function “grep” is used as in combination with regex-es, and based on the proposed regex, the command should return all the **lines** from the file which satisfy the regex.
22	grep regex file1 file2 file3	Multiple files (in this case file1, file2 and file3) can be listed as input to the “grep” function, and all of them are going to be searched in order to find the patterns proposed by the regex.

23	cat file1 file2 | grep ‘img[0-9]*\.jpg’	Regex in this expression is img[0-9]*\.jpg, which have to be written inside quotes. In this example the “grep” function is used to filter the input, meaning that it is for filtering the content of a file which was created a result of concatenation of file1 and file2.
24	grep ‘img[0-9]*\.jpg’ *.html	In this example command “grep” is not getting its input as a result of some commands and help of pipe symbols. Instead of that, all files of type html would be searched, because of the “*.html” part, to see if there is a matching line in the file which satisfies the regex. Instead of html we could have used any other extension which has text content, like for an example “.txt”.
25	command | grep -o ‘user=.*$’	By now we learned that command “grep” returns all the lines which satisfy the proposed regex, but it is sometimes the case that we do **not want to return the full lines, but only the matching part of the lines**. To do so we use a flag of “grep” command “-o”.
26	grep -v ‘^warning’ file	When we want to return only the **non-matching lines**, in contrast to what we done so far, we use the “grep” command and its flag “-v”.
27	cat file | grep -i ‘\.jpg’	If we want to perform a case-insensitive search (in contrast to the searches we have done above), we would use command “grep” and its flag “-i”.
28	grep -a regex binary_file	If we want to perform a search for matching lines with the regex of a binary_file, then we would use command “grep” and its flag “-a”.
29	egrep -a ‘(^Subject:)|(^From:)’ file	We use command “egrep” instead of “grep” when we want to use so called extended regex patterns, which include interpreting a pipe symbol inside a regex instead of an ordinary character as “or” symbol. So, for an example, if we wrote: `ls	grep ‘mp4|ovi’` the character ’|’ inside the regex will be interpreted as an ordinary character, and we would be looking for a whole string “mp4|ovi”. On the other hand, if we used command “egrap”, and write: `ls	egrep ‘mp4|ovi’` the character ‘|’ inside the regex will be interpreted as “or”. Using command “egrap” is equivalent to using command “grep” together with flag “-E”. By using command “egrep” or “grep -E” we are able to make multiple groups using regular expressions, like we did in the expression on the left. There are two types of groups we can use: **capturing-groups** and **non-capturing-groups**. While capturing-groups are used in this expression, when we want to have a non-capturing-group, we use the following syntax “`(?:red|green|blue)`“The question mark „?“ and the colon „:“ after the opening round bracket are the syntax that creates a non-capturing group.

30	cat file | sort	It performs a basic sorting of the content of the file, in an ascending order. You have to extra cautious when wanting to perform sorting, i.e. you have to be aware that this command looks at its input as string, and not numbers (unless instructed otherwise with use of a special flag), so if you use this command on number you may get a different output than you expected. For an example, the following set of numbers: {50, 5050, 5205, 51548970}, would not appear in the same order as in the set, but rather: 1. 50 2. 5050 3. 51548970 4. 5205 which may be contrary to expectations if you are not aware that the command looks at these values as strings and not numbers. By default, the „sort“ command sorts lines in an ascending order (meaning from 'a' to 'z', and '0' to '9', as numbers are here representing characters).
31	cat file | sort -n	If we want to perform sorting of inputs in a numeric manner, we would use command “sort” and its flag “-n”. By default, the „sort“ command sorts lines in an ascending order (meaning that in case we sort numbers it will sort from 0 upwards).
32	cat file | sort -r	By adding to command “sort” a flag “-r”, it will sort the input lines in a descending order (meaning from ‘z’ to ‘a’, and from ‘9’ to ‘0’).
33	cat file | sort -k number OR cat file | sort -t ‘ ’ -k number	If we want to perform sorting of all lines in a file based on a specific key in the lines, we use function “sort”, combine it with flag “-k” and afterwards write number, which specifies which key do we sort by. Note that number of the filed in a line starts from 1 (and not from 0 as someone could guess). The default separator is blank space, so the second command is equivalent to the first one.
34	cat file | sort -t ‘separator’ -k number	If in addition to specifying according to which key do we want to perform sorting of the lines of file, we can also specify the type of separator if it is different from blank space (which the default), because it is often the case that we have a file which has separated values with symbols such as commas, semicolon, pipe symbols, dash symbols etc. Example of separating content of each line of file “my_file”, which is written in such manner that all the fields in a line are separated by pipe symbols, based on the third key of in a line could be written as: cat my_file | sort -t ‘|’ -k 3
35	cat file | sort -u	If we want to perform sorting, but we do not want to get duplicates in the result we use command “sort” and use flag “-u” along it.

36	sort file | head -number	Command "head" can be used in combination with a number which comes after it like a flag, meaning that we use "-number" construction. This will return **first** (or from the top) number of entries, that is from the beginning of a file.
37	sort file | tail -number	Command "tail" acts in a same manner as command "head", except it returns number **last** entries from a file.
38	sort file | head -n -number	If we want to return all the entries from a file, **except** a certain number of entries which are **last** in the list, then we use command "head", in combination with flag "-n", which is followed by a "-number", where the number specifies how much of the **entries from the bottom will be ignored**.
39	sort file | tail -n +number	If we want to return all the entries from a file, **except** a certain number of entries which are **first** in the list, then we use command "tail", in combination with flag "-n", which is followed by "+number", where the number specifies how much of the **entries from the top will be ignored**.

40	sort file | uniq	**Eliminating duplicates**, making this expression equivalent to "sort -u file". This doesn't mean that lines which have duplicates are not displayed at all, but instead if a there are duplicates of one line, only one of such lines is going to be displayed.
41	sort file | uniq -c	Perform elimination of duplicates, but in addition, because of use of flag "-c", a number before each line is going to be displayed which indicated how much duplicates of a line exist in the file.
42	sort file | uniq -d	Displaying only lines which have duplicates, and ignoring lines which do not have duplicates.
43	sort file | uniq -u	Displaying only lines which do not have duplicates, and ignoring lines which have duplicates.

44	cat file | cut -f 1,3	In this expression the flag "-f" stands for "**field**". The command is used to only **return values from specific fields**, which we specify by writing number after the "-f" flag separated with commas.
45	cat file | cut -f 2-4	Another way in which we can specify to only return values of specific fields is by using the flag "-f", but then instead of writing the numbers of fields which values we want to return separated by commas, we could simply use syntax "number1-number2", after which all fields which have index in range [number1, number2] are going to be returned.
46	cat file | cut -f 2-4 -d ','	Also, we can specify the separator of the file we want to extract values from by using flag "-d" (which is short for "**delimiter**"), after which we put the separator in quotation marks.

47	cat file | cut -c 1-10	This approach is very dangerous unless you know the exact structure of the file you want to perform analysis on. The reason why is it dangerous is because by using command “cut”, in combination with flag “-c”, which is followed by a range of values, we specify that we want to perform cutting of file lines by 10 characters. Example when this could be used is when we are working with car table plates for a specific country, where each table plate has to have a specified structure, i.e. number of characters.
48	cat file | wc	Command “wc” returns three values for an input file and these are: 1. line count 2. word count 3. character count
49	cat file | wc -l	Command “wc” followed by a flag “-l” returns number of lines in an input file.
50	cat file | wc -c	Command “wc” followed by flag “-c” returns number of characters (i.e. bytes) in an input file.
51	cat file | shuf	Command “shuf” is used to shuffle the content of file.
52	cat file | shuf | head -10000	The reason why “shuf” command is very useful is because we can use it in combination with command “head”, or command “tail” if we have a huge file filled with data to get an idea about the data that is saved inside the in the huge file. This is in some sense a **random sample** taken from a huge dataset, which helps us get a more realistic picture of the content of the file as we are taking the sample from.

53	cat file | sed ‘s/oldText/newText’	Command “sed” is short for “**stream editor**”. Character ‘/’ is used to separate parts of the quoted part of the expression. Note that use of slash as character which will separate the quoted part of the expression in not mandatory, and you can instead use any character you like. For an example you can use semicolon character as separating character, and rewriting the expression from the left we could equivalently write it as: cat file | sed ‘s;oldText;newText’ The first part of the quoted part of the expression is character ‘s’, which stands for “**substitute**”, while the second part is the oldText, which is the text which will be replaced with the third part of the quoted part of the expression, newText. Such use of command “sed” will **only replace the first occurrence** of oldText with newText.
54	cat file | sed ‘s/oldText/newText/g’	This expression differ from the above one only by the fact that in the quoted part of the expression we have one additional, fourth part which is character ‘g’, which stands for “**globally**”. Such use of command “sed” **will replace all the occurrences** of oldText with newText.
55	cat file | sed ‘s/img_\([0-9]*\).jpg/image\1/g’	As we know from regular expressions, the content inside rounded brackets is content which will be extracted for further processing, and this is used in the second part of the quoted part of the expression, while in the third part we reference this extracted content with “\1” construct. We can see that in the regex before each of the brackets we had to use a slash character ‘\’, which was obligatory if we wanted the brackets to be interpreted in a way of extracting part of a regex.
56	cat file | sed ‘s/file[0-9]*.png/”&”/I’	We can see that the fourth part of the quoted part of the expression is upper case letter “I”. By default, command „sed“ is case-sensitive, but you can add “I” flag at the fourth part of the quoted part of the expression to change this, that is to perform a **case-insensitive search**.
57	cat file | sed -e ‘s/old1/new1/’ -e ‘s:a/b:a_b’	We can also specify multiple patterns according to which a search of the input file is going to be performed, but in this case in from of each of the patterns we want to specify we must write flag “-e”.
58	sed -i ‘s/oldText/newText’ file	By using command “sed”, and its flag “i” we would do the replacement we intended not in a manner that it would just simple be displayed in the terminal and not change the real content of the file, but quite opposite. Nothing would be outputted in the terminal after this command id executed, but the real content of the file would be modified.

59	join -1 2 -2 1 file1 file2	We should look at part of this expression at a time to better understand it. Firstly, we observe that we are working with two files and by using the command "join" we can guess that we perform a join on these two files. The fields of the two files according to which we perform this join have to be specified in such an expression. In the example, by looking at "-1 2" part of the expression we conclude that from the file which is listed first in this expression (which is file1) we use the 2 field for performing the join, while on the other hand by looking at the "-2 1" part of the expression we conclude that from the file which is listed second in this expression (which is file2) we use the 1 field to perform the join. This kind of joining would be equivalent to an **inner join**.
60	join -1 2 -2 1 -t ',' file1 file2	If in addition to performing the join we want to specify the separator which is used in the files we want to join, we use flag "-t", followed by a separating character in quotation marks.
61	join -1 2 -2 1 -a 1 file1 file2	If in contrast to previous two expression we would like to make a join which is equivalent to an **outer join**, we could use an expression as this one, in which we used flag "-a" followed by "1". This specifies that we want to keep even the lines which are not matching.
62	join -v -1 2 -2 1 file1 file2	

- **The join commands will only work if the input file(s) it gets is sorted according to the filed we want to perform a join on.**
- If you do some heavy use of join commands and more advanced operations you should consider moving to more powerful tools than Bash commands.

63	awk 'BEGIN {init-code} {per-line-code} END {done-code}' file	Command "awk" is used for executing command on each line of code of the input. To be more specific init-code is the code which is executed before look at any line of the input file, while code per-line-code is execute for each line of the input file. At the end code done-code is executed, and it gives the final output of such an expression.
64	awk 'BEGIN { x=0 } { x=x+1 } END { print x }'	This expression is used if we want to count number of lines in the input file.
65	awk 'BEGIN { x=0 } { x=x+$2 } END { print x }'	In the previous expression we simply counted the number of lines in a file without referencing any field of the input file. Here, on the other hand we do reference part of each of the lines, and it is the second field of each of the lines from the input file, and it is done by using "$2". In this expression we sum up all the second fields in the file.
66	awk 'BEGIN { x=0; y=0 } { x=x+$2; y=y+1 } END { print x/y }'	In this expression it is shown that we can use multiple lines of code in each of the code parts of the expression.
67	awk 'BEGIN { x=0 } { if($1 > 10) x=x+$2 } END { print x }'	In this expression it is shown that we can use conditions inside any of the code parts of the expression.

68	awk -F '|' 'BEGIN { s=0 } { s=s+$4 } END { print s }'	When we want to specify a separator for "awk" command we use its "-F" flag, which should be followed by the separator in quotation marks, which specifies how is the data separated in the input file.
- For more in depth knowledge of "awk" command you can refer to webpage: https://learnxinyminutes.com/docs/awk/		

Examples from lectures:

1. Find all senders on the LKML (Linux kernel mailing list).

```
xzcat file.xz | grep -a '^Sender'
```

2. Find all senders that are not using „.com“ domains.

```
xzcat file.xz | grep -v '^Sender .* \.[^c][ ^o][ ^m]'
```

3. Extract the email part of the sender address.

```
xzcat file.xz | grep '^Sender ' | grep '<([^>]+)>$'
```

4. Search for a date which is not in October.[1]

```
xzcat file.xz | grep '^Date' | grep -v 'Oct'
```

5. Find all posters to the LKML, sorted alphabetically.

```
xzcat file.xz | grep '^Sender' | grep -o ' [^<]*' | sort -u
```

6. Find the 10 most prolific (present in large numbers or quantities) posters to the LKML.

```
xzcat file.xz | sed ‘s;’^Sender: \( .*\)’; \1;g’ |uniq -c | sort -n -r | head -10
```

7. Find all posters who wrote only a single mail.[2]

[1] lecture slides „Basic Building Blocks“, slide 12

```
xzcat file.xz | sed ‘s;’^Sender: \( .*\)’; \1;g’ |uniq -u
```

8. Count the number of mail in the archive.

   (I could not find “archive” in the file)

9. Pick 20 mail subjects at random.

```
xzcat file.xz | sed ‘s;’^Subject: \( .*\)’; \1;g’ | shuf | head -20
```

10. Compute the most popular content type.[3]

```
xzcat file.xz | sed ‘s;’^Content-Type: \( .*\)’; \1;g’ | uniq -u | sort -n -r | head -1
```

11. Find the most popular words in the LKML data set.

```
xzcat file.xz | sed 's;[^A-Za-z];\n;g' | grep -v '^$' | sort | uniq -c | sort -u | tail -10
```

12. Compute the average number of lines in a mail.

```
xzcat file.xz | grep '^Lines:' | awk -F ' ' 'BEGIN{sum=0;count=0} {count=count+1; sum=sum+$2} END{print sum/count}'
```

13. Compute median (not average) of lines in a mail.

```
xzcat file.xz | grep '^Lines:' | wc -l     #the result is 10025 and we will use output of this command for the next line of code

xzcat file.xz | grep '^Lines:' | sort -k 2 -n | awk -F ' ' 'BEGIN{count=0; median=0} {count=count+1; if(count == 10025/2) median = $10025/2}
                                                                      END{print median}'
```

14. Find all mail authors who never get a response.[4]

[2] lecture slides „Basic Building Blocks“, slide 16
[3] lecture slides „Basic Building Blocks“, slide 20

15. Count the sum using „awk“ command.[5]

```
time awk -F '|' 'BEGIN{x=0} {x=x+$5} END{print x}' lineitem.tbl
```

[4] lecture slides „Basic Building Blocks“, slide 24
[5] lecture slides „Basic Building Blocks“, slide 28

Examples from exercise sheets:

1. [exercise sheet 1, exercise 5] Write a regular expression that matches all e-mail addresses in the file mails_match.txt and that does not match any of the lines in mail_dont_match.txt.

    ```
    egrep ’^[a-zA -Z0-9._%+ -]+@[a-zA-Z0 -9. -]+\.[a-zA-Z]{2 ,}$’ mails_match .txt
    ```

2. [exercise sheet 1, exercise 6, problem a)] Use non-capturing groups to find a short expression to match MAC addresses in the file random-mac.txt. Note that „grep“ needs the flag „-E“ to support groups.

    Firstly we have to know that MAC addresses consist of 12 hexadecimal numbers which are grouped in two digits, meaning that we have 6 groups consisting of 2 hexadecimal numbers. The „short extension of a MAC address“ in the problem setting refers to the first 5 groups of 2 hexadecimal digits.
    Soulution in which a dash character is used between the 6 groups:

    ```
    egrep '(?:[A-F0-9][A-F0-9]-[A-F0-9][A-F0-9]-[A-F0-9][A-F0-9]-[A-F0-9][A-F0-9]:)-[A-F0-9][A-F0-9] ' random-mac.txt
    ```

    or an equivalent solution (in which we supposed a dash was not used between 6 groups of hexadecimal numbers):

    ```
    egrep '(?:[A-F0-9]{2}:){5}[ A-F0-9]{2}
    ```

3. [exercise sheet 1, exercise 6, problem b)] Use capturing groups and replace to change the structure of the file names.txt from ”Firstname Lastname” to ”Lastname, Firstname”.

    ```
    cat names.txt | sed ‘s;^\([A-Za-z]*\) \([A-Za-z]*\)$;\2 \1;g’
    ```

4. [exercise sheet 1, exercise 7] Convert RGBA color codes from the hex form ”#FF0000FF” (red) to the rgba form ”rgba(255, 0, 0, 1.0)”. In the rgba form the first three digits indicate how much of the additive light basic colors have to be added to generate the color. The fourth entry determines the alpha value, also known as opacity which ranges from 0 to 1. In the hex form two hexdigits represent the values of red, green, blue and alpha, all with a range from 0 to 255.

    As name of the file from which we should convert the hex form of colors into RGBA notation, we will assume that this file is called „hex_colors“.

Additionally, the description of how to convert a a hex color into RBGA notation is a little bit messy in the problems setting, put in short, first two (the left-most) hex digits denote the opacity, which we will note with „O“, the following two hex digits denote color red (we will note it with „R“), and the following two digits describe color blue (we will denote it with „B“) are, and the last two are describing color green (we will denote it with „G“). Now, the RGBA notation is given as in the following syntax: „rgba(R, B, G, O)“, where each of the {R, G, B, O} are not hexadecimal numbers, but decimal numbers.

```
cat hex_colors.txt | sed ‘s;^#\([A-F0-9]{2}\)\([A-F0-9]{2}\)}\)\([A-F0-9]{2}\)}\)\([A-F0-9]{2}\);\2, \3, \4, \1;g’ |

          awk -F ‘,’ ‘BEGIN{R=0; G=0; B=0; O=0;} { R=toDeciaml($1); G=toDeciaml($2); B=toDeciaml($3); O=toDeciaml($4)} END{ print
                    “rgba(R,G,B,O)”}’
```

The function “toDecimal(hex_number)” actually does not exist, but as writing such a message is not the point of this problem, an imaginary function is used in order to keep the solution short and keep the focus on the use of GNU bash command.

5. [exercise sheet 2, exercise 2, problem 1] Answer the following query for the Linux Kernel Mailing List data set (the file containing this dataset is called „gmane.linux.kernel“) using GNU utils and bash functionality: Of how many lines does the longest mail consist? Challenge: Compute the number of lines in each mail without using the ’Lines:’ attribute in the dataset.

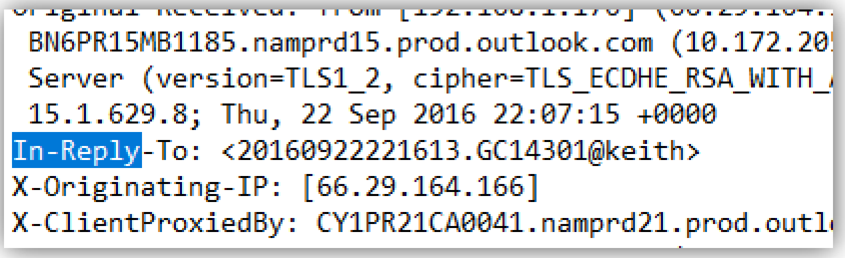
   As we have no idea how the content of the file “gmane.linux.kernel” look like, here is a short explanation after I took a quick look. Basically, the file contains multiple mails, listed one below another. As the “challenge” part of the problems setting says, there is heading in each of the mails which say how much lineas a mail contains. So, we are firstly going to provide an answer which takes advantage of the line of the mail which says how many lines does a mail consist of, and afterwards we will give a solution as if this info did not exist in each mail.

   ```
   egrep ‘^Lines: ([0-9]*)’ gmane.linux.kernel | sort -r -n | head -1
   ```

   Now we should count the number of lines in each mail without taking advantage of the fact that there is a line each of the mails which outputs how many lines that mail has, but it is very unclear what should be counted as a line in the document, so we will not do this task.

6. [exercise sheet 2, exercise 2, problem 2] Answer the following query for the Linux Kernel Mailing List data set (the file containing this dataset is called „gmane.linux.kernel“) using GNU utils and bash functionality: Which mail received most (direct) replies?

A mail received a direct reply if the mail address is in “In-Replay-To: ” part of some other email, and after this string comes email in form of “<email_address>”, like in the photo below:

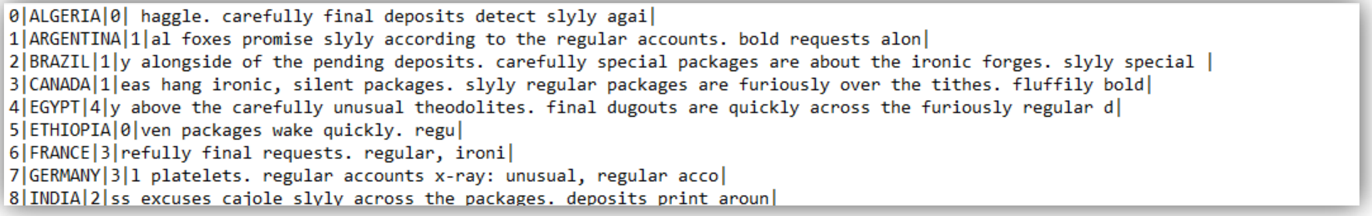


```
 BN6PR15MB1185.namprd15.prod.outlook.com (10.172.20
 Server (version=TLS1_2, cipher=TLS_ECDHE_RSA_WITH_
 15.1.629.8; Thu, 22 Sep 2016 22:07:15 +0000
In-Reply-To: <20160922221613.GC14301@keith>
X-Originating-IP: [66.29.164.166]
X-ClientProxiedBy: CY1PR21CA0041.namprd21.prod.outl
```

cat gmane.linux.kernel | sed ‘s; ‘^In-Reply-To: <\(.*\)>$’;\1;g’ | uniq -c | sort -n -r | head -1

7. [exercise sheet 2, exercise 3, problem 1] Answer the following queries for the TPC-H data set (in this task out of all files we use „nations.tbl“ file and “region.tbl” file) using GNU utils and bash functionality. You can download the data set using the links provided in the Moodle course and this link[6] contains the schema. According to the data set, which nations are in Europe?

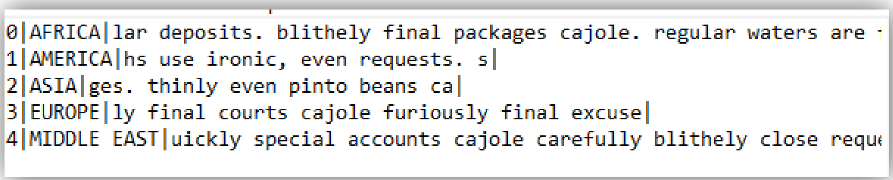
The “nations.tbl” file has the following structure:



```
0|ALGERIA|0| haggle. carefully final deposits detect slyly agai|
1|ARGENTINA|1|al foxes promise slyly according to the regular accounts. bold requests alon|
2|BRAZIL|1|y alongside of the pending deposits. carefully special packages are about the ironic forges. slyly special |
3|CANADA|1|eas hang ironic, silent packages. slyly regular packages are furiously over the tithes. fluffily bold|
4|EGYPT|4|y above the carefully unusual theodolites. final dugouts are quickly across the furiously regular d|
5|ETHIOPIA|0|ven packages wake quickly. regu|
6|FRANCE|3|refully final requests. regular, ironi|
7|GERMANY|3|l platelets. regular accounts x-ray: unusual, regular acco|
8|INDIA|2|ss excuses cajole slyly across the packages. deposits print aroun|
```

[6] https://hyper-db.de/tpch.png

while on the other hand “region.tbl” file has the following structure:



```
0|AFRICA|lar deposits. blithely final packages cajole. regular waters are
1|AMERICA|hs use ironic, even requests. s|
2|ASIA|ges. thinly even pinto beans ca|
3|EUROPE|ly final courts cajole furiously final excuse|
4|MIDDLE EAST|uickly special accounts cajole carefully blithely close requ
```

As we seen the content of both files we can conclude that the third entry in “nations.tbl” represents the key for the table “region.tbl”, and therefore the continent on which a country is located on. By observing these two tables we can conclude that the countries with key “3” are located in Europe (or by simply stating that countries “FRANCE” and “GERMANY” are located in Europe). In this case we would simply write:

```
cat nations.tbl | cut -f 2,3 | grep ‘|3|$’
```

or if we want to output just the names of the countries:

```
cat nations.tbl | cut -f 2,3 | grep ‘|3|$’ | cut -d ‘|’ -f 1
```

But, if we have not seen the content of these two files and only know what fields should be paired in order to conclude which country is in which continent, then in order to get an answer we would need to perform a join between two tables (Also take into consideration that some entries in the tables could be entered wrongly, like for an example if someone accidently added “FRANCE” and “GERMANY” to “ASIA”). Then the following code could be used for solving the problem:

```
join -1 3 -2 1 -t ‘|’ <( sort -t ‘|’ -k 3 nations.tbl) <(sort -t ‘|’ -k 1 region.tbl) | grep ‘EUROPE’ | cut -d ‘|’ -f 3
```

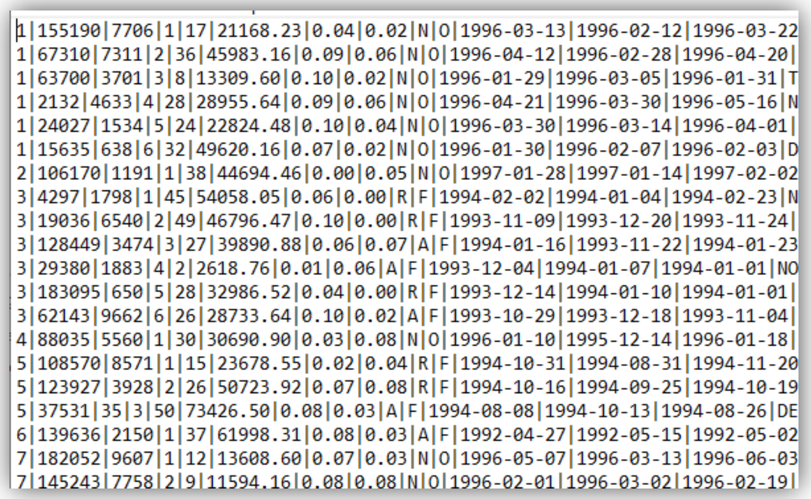
is the correct command for answering the problems, if we know that the result of the join will be such an output in which name of a country is in the third position.

8. [exercise sheet 2, exercise 3, problem 2] Answer the following queries for the TPC-H data set (in this task out of all files we use „lineitem.tbl“ file) using GNU utils and bash functionality. You can download the data set using the links provided in the Moodle course and this link contains the schema. How many lines are in lineitem.tbl?

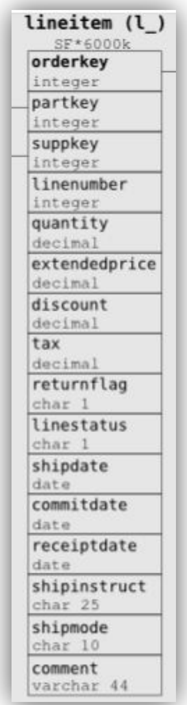
```
wc -l lineitem.tbl
```

9. [exercise sheet 2, exercise 3, problem 3] Answer the following queries for the TPC-H data set (in this task out of all files we use „lineitem.tbl“ file) using GNU utils and bash functionality. You can download the data set using the links provided in the Moodle course and this link contains the schema. Determine for each part how often it has been bought. (Hint: Sum up the quantities in „lineitem.tbl“, grouped by „partkey“).

To see what columns represent we take a look at the schema, and take a look as well at the content of the „lineitem.tbl“ file:



```
1|155190|7706|1|17|21168.23|0.04|0.02|N|O|1996-03-13|1996-02-12|1996-03-22
1|67310|7311|2|36|45983.16|0.09|0.06|N|O|1996-04-12|1996-02-28|1996-04-20|
1|63700|3701|3|8|13309.60|0.10|0.02|N|O|1996-01-29|1996-03-05|1996-01-31|T
1|2132|4633|4|28|28955.64|0.09|0.06|N|O|1996-04-21|1996-03-30|1996-05-16|N
1|24027|1534|5|24|22824.48|0.10|0.04|N|O|1996-03-30|1996-03-14|1996-04-01|
1|15635|638|6|32|49620.16|0.07|0.02|N|O|1996-01-30|1996-02-07|1996-02-03|D
2|106170|1191|1|38|44694.46|0.00|0.05|N|O|1997-01-28|1997-01-14|1997-02-02
3|4297|1798|1|45|54058.05|0.06|0.00|R|F|1994-02-02|1994-01-04|1994-02-23|N
3|19036|6540|2|49|46796.47|0.10|0.00|R|F|1993-11-09|1993-12-20|1993-11-24|
3|128449|3474|3|27|39890.88|0.06|0.07|A|F|1994-01-16|1993-11-22|1994-01-23
3|29380|1883|4|2|2618.76|0.01|0.06|A|F|1993-12-04|1994-01-07|1994-01-01|NO
3|183095|650|5|28|32986.52|0.04|0.00|R|F|1993-12-14|1994-01-10|1994-01-01|
3|62143|9662|6|26|28733.64|0.10|0.02|A|F|1993-10-29|1993-12-18|1993-11-04|
4|88035|5560|1|30|30690.90|0.03|0.08|N|O|1996-01-10|1995-12-14|1996-01-18|
5|108570|8571|1|15|23678.55|0.02|0.04|R|F|1994-10-31|1994-08-31|1994-11-20
5|123927|3928|2|26|50723.92|0.07|0.08|R|F|1994-10-16|1994-09-25|1994-10-19
5|37531|35|3|50|73426.50|0.08|0.03|A|F|1994-08-08|1994-10-13|1994-08-26|DE
6|139636|2150|1|37|61998.31|0.08|0.03|A|F|1992-04-27|1992-05-15|1992-05-02
7|182052|9607|1|12|13608.60|0.07|0.03|N|O|1996-05-07|1996-03-13|1996-06-03
7|145243|7758|2|9|11594.16|0.08|0.08|N|O|1996-02-01|1996-03-02|1996-02-19|
```



A group by can me mimicked by using firstly command “sort” and then using command “awk”.

```
cat lineitem.tbl | sort -t ‘|’ -k 2 | awk -F ‘|’ ‘BEGIN{ partkey=0; sum=0} { if($2 > partkey){print partkey; print sum; sum=0; partkey=$2};
sum=sum+$5} END{ print partkey; print sum }’
```