Regular expressions

This step goes further into working with text files by introducing regular expressions.

- Filtering
- Introduction to regular expressions
- Constructing regular expressions
- Summary and test

Filtering

In this section we will use the files results1.txt and results2.txt that contain lists of results for made-up persons. Save these files and use them for the exercises further down.

The grep command is used to show all lines of a file that contains a certain string of characters. We can search for the string *Johan* in the first file with this command:

zaza12 <1> **grep Johan results1.txt**
Johanna Rönnberg 9 15 13
Göran Johansson 16 19 15
Yngve Johanson 10 11 8

All lines containing the text *Johan* are displayed, but the result was perhaps not exactly what we wanted. The grep command will find all occurrences of *Johan*, also when it is part of another name. If we want to search for just the name Johan we can for example search for "Johan " with a space after the last letter. We must then put quotes around the argument to grep.

Another way of finding only the name Johan is to use the flag -w. There are many flags that modify the behaviour of grep. The table shows some of the most useful.

Flag	Function
-i	Ignore upper case and lower case
-v	Search for lines that do not match the supplied word
-w	Search for the string as a word, e g there has to be spaces around it
-n	Display the row number for all matching rows
-c	Only display the total number of matching rows, not the rows themselves

With grep and these flags we can answer different questions about our files. Which persons have a name including the string *lind* and what rows are they on?

zaza12 <2> **grep -i -n lind results1.txt**

18:Ingrid Lindgren 8 13 13
21:Elisabeth Björklind 15 12 10

We can search sveral files at the same time by adding them as arguments to grep. What persons, in both files, have the result 17 in any table?

zaza12 <3> **grep 17 results1.txt results2.txt**
results1.txt:Mikaela Andersson 14 - 17
results1.txt:Annica Rundgren 17 10 15
results1.txt:Mats Karlzon 15 17 10
results2.txt:Lina Larsson 13 17 7
results2.txt:Ann Nordenberg 17 14 14

How many persons in the second file has a result in all three tables, that is, the text " -" is not on the line?

zaza12 <4> **grep -c -v " -" results2.txt**
19

Note that we added a space before the hyphen to avoid matching hyphenated names such as Jan-Erik or Ellis-Bextor.


Exercise

(1) How many persons in the first file have names ending with *son*?

Introduction to regular expressions

With **regular expressions** we can use grep to search for more complex patterns. How would you search for all possible spellings of the Swedish last name Karlsson? It can be spelled with a C or a K, with one s or two, and sometimes even with a z. By creating a regular expression we can match all these with a single search. We will show how later in this section.

Regular expressions are similar to the wildcards we used to search for multiple files in the terminal, but regular expressions are more powerful. Regular expressions are used not only as search patterns to grep but also appears in other corners of UNIX, and in most programming languages. It is also one of the basics for the study of formal languages. Therefore it is useful for almost everyone who works with computers to know about regular expressions.

When we use regular expressions to search with grep, the idea is that you can use certain special characters to create a pattern that can match several words or text strings. The simplest regular expressions consist only of letters and we have actually seen them already. If you search for *abc* you will find lines that contain the string *abc*.

Single characters

To search for patterns that match arbitrary characters, you can add a dot to the regular expression. A dot matches any character. To search for all occurences of two *e* with exactly one arbitrary character between, we can do like this:

```
zaza12 <5> grep e.e results2.txt
Peter Bajramovic    20   9  12
Johan Seeger        13  16   9
Therese Nilsson     15  15   7
```

If we want to search for specific characters, we can supply them within square brackets [...]. Each set of square brackets matches exactly one of the characters supplied between the brackets. We can search for all strings that start with uppercase or lowercase *A* or *E*, followed by *nn* and finally either *a* or *e* like this:

```
zaza12 <6> grep "[AaEe]nn[ae]" results2.txt
Annette Sundman     20  19  21
Johanna Tuhkunen    22   9  11
Hannes Olsson       16   8  10
Johanna Dahlerup     7  10  22
Anneli Eliasson      8  22  11
```

We have to add qoutes to the search pattern to avoid that the terminal misinterprets the brackets. We can also put intervals of characters between the brackets. The pattern [a-k] matches all lowercase letters from *a* to *k*. However, letter intervals may vary depending on your language settings, so take care in using them. We can also use number intervals, for example to find all results between 20 and 29 like this:

```
zaza12 <7> grep "2[0-9]" results2.txt
Peter Bajramovic    20   9   12
Per Karlsson        21   -   21
Engla Hedberg       21  11   21
...
```

Exercise

(2) Which persons, in both files, have name containing *kvist* or *qvist*?

Exercise

(3) Which persons, in both files, have only odd results in all three columns?

Constructing regular expressions

This section will go through how to create regular expressions to search for more complex patterns.

Repeats

By putting a star * after a part of a regular expression, you allow that part to occur any numer of times (including zero). The star can for example be put after a letter, a dot or a set of square brackets. We can search for names that have uppercase *A* followed by any numer of arbitrary characters and then lowercase *n* like this:

```
zaza12 <8> grep "A.*n" results2.txt
Annette Sundman     20  19  21
Jonas Andersson     11  12  12
Ann Nordenberg      17  14  14
Arten Wärn          19  10   -
Anneli Eliasson      8   22  11
```

We can also solve the problem stated earlier, matching all different spellings of Karlsson:

```
zaza12 <9> grep "[CK]arls*z*on" results1.txt results2.txt
results1.txt:Gustaf Karlsson     12  14  13
results1.txt:Marcus Carlsson     12   9  12
results1.txt:Josefine Carlson    18  14  16
results1.txt:Mats Karlzon        15  17  10
results2.txt:Per Karlsson        21   -  21
```

We search for sequences that start with either *C* or *K*, continues with *arl*, any number of *s*, any number of *z* and finally *on*.

Anchoring

To search for something specifically at the start or end of a line we can use the anchors ^ or $. We can search for all people whose first names begin with *E* like this:

```
zaza12 <10> grep ^E results2.txt
Engla Hedberg       21  11  21
Erik Hernqvist       7   7   9
```

Without the anchor, we would also have gotten all people whose last names start with *E*. By combining the anchors we can search for complete rows. For example we can count all empty rows like this:

```
zaza12 <11> grep -c ^$ results2.txt
0
```

Alternatives

So far we have only covered **basic** regular expressions. There is also an **extended** version. Normally you'll do fine with just the basic set, but one interesting aspect of the extended version is an easy way to express alternatives.

To use extended regular expressions, you have to use the egrep command. It works like grep, only with more options for constructing regular expression. Different options are separated by the vertical bar |. We can search for people with either a first name beginning with *S* or a last name beginning with *F* like this:

```
zaza12 <12> egrep "^S| F" results1.txt
David Fors         8  12  11
Siv Tidblom       11   9  12
```

Exceptions

What if you want to search for one of the characters that are used to construct regular expressions? If those characters are to be interpreted literally, you have to put a backslash before them. To search for *, type *.

Exercises


Exercise

(4) Which persons, in both files, have names ending in *a*?


Exercise

(5) Which persons, in both files, have a first name with exactly three letters?


Exercise

(6) How many persons, in both files, have the first name Johan and have results in all three columns?

Pattern	Explanation
.	matches exactly one arbitrary character
[ ]	matches exactly one of the characters within brackets
*p**	matches the pattern *p* zero, one or more times
^	matches the beginning of the row
$	matches the end of the row
p1|*p2*	matches either of patterns *p1* and *p2* (only works with egrep)
c	matches the character *c* if *c* is one of the special characters used to construct regular expressions

These flags can be used with grep:

Flag	Function
-i	Ignore upper case and lower case
-v	Search for lines that do not match the supplied word
-w	Search for the string as a word, e g there has to be spaces around it
-n	Display the row number for all matching rows
-c	Only display the total number of matching rows, not the rows themselves

Test

1. Give a regular expression matching one of the number 2 to 4!

2. Give a regular expression matching lines that contain only numbers!

The following questions assume that you have the file list2.txt available. To answer the questions, you have to search through the file and input the number of matches here. (Please note that the last name 'Lind' ends with 'lind' as well.)

3. How many people have a last name ending with stedt?

4. How many people have the first name Erik or Eric?

5. How many women are there (ninth number in personal number is even)?

Cutting, pasting and redirecting

One of the strengths of UNIX is its ability to process text files in different ways. There are many commands and techniques that can be used for this. Studying these gives insight into the UNIX philosophy. In this step we will show how to save program output to a file, how one commands's output can be another one's input, and different ways to cut and paste in textfiles.

- Introduction to redirecting
- Sorting
- Cutting and pasting
- Summary and test

Introduction to redirecting

The UNIX shell doesn't get input straight from the keyboard, and neither does it write output straight to the screen. In- and output are done via two streams called stdin (*standard input*) andstdout (*standard output*). Normally, these are connected to the keyboard and the screen respectively, but they can be connected elsewhere. For example, you can let a program get its input from a file rather than from the keyboard. This is called **redirecting** and is an important UNIX feature. Several simple commands can be conjoined, and so rather complex problems can be solved easily. The table gives a brief overview of redirecting:

command > file	Send output from a program to a file
command >> file	Send output from a program to the end of an existing file
command < file	Get input from a file
command1 | command2	Use output from one program as input for another

We will now give some examples of how these commands are used.

Write to a file using > or >>

By typing *command > file*, the output of the command will be redirected to a file. For example, to get the output of the ls command saved to a file, type:

ls > files.txt

When you use redirecting as in the example above, you will get an error message if the file files.txtalready exists. If you want to add to the end of the file instead, use >>. This way, files can be added to the end of other files, for example like this:

cat latest_results.txt >> all_results.txt

The cat command outputs the contents of a file, but the output is redirected and added to the end of the file all_results.txt.

You can force an existing file to be overwritten by using e.g.

ls >! files.txt

Combining commands using |

If you want to redirect one program's output to become input to another program, use the vertical bar |. The bar is also called pipe, and you can imagine it connecting two commands, letting data flow through the pipe. Instead of typing ls -l *.txt you can type

ls -l | grep ".txt"

which will list all lines in the output from ls -l that contain the text .txt. The word *pipe* is also used as a verb, i e what we just did was *piping* ls to grep. The grep finds all lines in a file that contain a certain text string. We will cover grep more in the next step.

Pipe can also be combined with other redirections. Suppose that you have a file random_words that contains 1000 words, one on each line. You want to sort these alphabetically and save them in the file sorted_words. This can be done using the sort command like this:

cat random_words | sort > sorted_words

The cat random_words command sends output (i e the contents of the file) as input to the sort command. The output from sort is then sent to the file sorted_words.

The following section will introduce some commands that are useful to redirect in- and output.


Exercise

(1) How can you use redirecting to find all occurrences of both words "milk" and "cookies" on the same row in a file?


Exercise

(2) The flags -F or -p attached to the ls will mark all directories in a certain way. How can you make a list of all directories, using what you have learnt so far?

Sorting

UNIX commands that edit text files often work row by row. The sort command is used to sort the rows in a file. As an example, we will use the file results1.txt.

If sort is given no other arguments than which file to sort, this is the result:

```
zaza12 <1> sort results1.txt
Annica Rundgren     17  10  15
Britt Lidell         -  14   9
David Fors           8 12  11
Elisabeth Björklind 15  12  10
Gustaf Karlsson     12  14  13
...
```

With different flags, it is possible to sort in many different ways. The table below summarizes some of the most important flags:

Flag	Function
-r	Sort backwards
-k *n*	Sort by the n:th field of the row
-t *c*	Use *c* as a separator between fields
-b	Ignore spaces at the beginning of the line
-n	Sort numerical values

The sort command can sort by special parts of a line called *fields*. The fields are normally separated by spaces, so in our example file, the last names are the second field. If the fields are separated by other characters, this can be marked with the -t flag. To sort our example file backwards by last name, we do like this:

```
zaza12 <2> sort -r -k 2 results1.txt
Siv Tidblom         11   9  12
Johanna Rönnberg     9  15 13
Annica Rundgren     17  10 15
Ingrid Lindgren      8  13  13
Britt Lidell         -  14   9
Jerker Leo           -   -  11
...
```

If we don't supply sort with a file name, input will be taken directly from stdin. This means that sort can easily be connected with other commands. Suppose that we wish to produce a list of all files in a certain directory, ordered by size. The command ls -l gives us a detailed list of the directory contents, where file size is the fifth field. We can send output from ls to sort like this:

```
zaza12 <3> ls -l | sort -k 5
total 350
-rw-r--r--   1 petda123 student      744 aug 17 15:07 notes.txt
-rw-r--r--   1 petda123 student     2541 jun  30 15:09 letter.txt
-rw-r--r--   1 petda123 student     2849 dec 12 15:09 database.mbd
```

-rw-r--r-- 1 petda123 student 160968 jun 30 15:08 thesis2.doc

Sorting in the example above is easy, since the file size is aligned to the right. If the file we wish to sort looks like results3.txt and we wish to sort by result, we must do it this way:

```
zaza12 <4> sort -n -b -k 2 results3.txt
 Per-Åke  918
  Therese  992
    Jörgen  1012
      Lisa  1320
     Sofie  1439
     Kalle  1745
```

Here -n is used to specify that we want to sort by numbers, not in alphabetical order, and -b to ignore extra spaces between fields.


Exercise

(3) Sort the file results4.txt by date of birth (first six numbers in the third field).


Exercise

(4) Sort the file results5.txt backwards by the results in the first column.

Cutting and pasting

This section covers four commands used for cutting and pasting in text files.

The cut command

The cut command can be used to cut one or more columns from a text file. The result will have just as many rows, but only the chosen columns will be included. For example, we can cut all first names from the file results4.txt like this:

```
zaza12 <5> cut -f 2 -d : results4.txt
Johan
Tommy
Olof
Jesper
...
```

The most important flags to cut can be found in this table:

Flag	Function
-c *spec*	cuts the characters specified by *spec* from each row
-f *spec*	cuts the fields specified by *spec* from each row
-d *c*	use the character *c* as separator between fields

Both flags -c and -f ar used in the same way. To cut a single character or field,, specify a number. To cut several characters or fields, specify numbers separated by comma or an interval with a dash. Our example file starts with a field that looks like student ID. We can cut the numbers from this by specifying 6-8 like this:

```
zaza12 <6> cut -c 6-8 results4.txt
101
102
103
104
...
```

The paste command

The opposite of cut is paste. Using paste, several files can be joined to one by considering the files columns. Normally the columns are pasted together using tab as a separator, but with the-d flag, one or more characters to use as a separator can be specified.

The files results5.txt and results6.txt contain two lists with the same people. They can be pasted together like this:

```
zaz12 <7> paste results5.txt results6.txt
Joanna Johansson    20  36    Joanna Johansson    39  42  24
Patrik Aspebring    26  37    Patrik Aspebring    28  23  33
Timmy Sallnäs       38  29    Timmy Sallnäs       36  41  25
...
```

The names in both lists are the same. To just get the name once, we can combine paste with cut to cut the columns we are interested in:

```
zaza12 <8> paste results5.txt results6.txt | cut -c 1-27,48-
Joanna Johansson    20  36  39  42  24
Patrik Aspebring    26  37  28  23  33
Timmy Sallnäs       38  29  36  41  25
...
```

The head and tail commands

The head command will cut the first lines from a file and tail will cut the last ones. These can be used to have a quick look at the beginning or the end of a large file. But sometimes they can be useful combined with other commands.

Both commands will return ten lines unless something else is specified. To cut a certin number, use the flag *-n* where n is the number of lines to cut. The tail command can also be used with the flag *+n*, which means that all lines from and including line *n* will be cut.

```
zaza12 <9> head -2 results5.txt
Joanna Johansson     20  36
Patrik Aspebring     26  37
```


Exercise

(5) Start with the file results4.txt. Make a new list that contains only first names and the three columns of results.


Exercise

(6) Start with the file results4.txt. If the list is sorted by first name, what is the first name of the first person? Solve the problem by combining cut, sort and head on one line.

Counting occurrences

The grep command can be used to search in a text file. The command goes through the file line by line and for each line checks whether it contains the search phrase. Normally grep prints all matching lines, but with the flag -c, occurrences will only be counted.

The example below counts the lines in the file resultat5.txt that contain the text string "son".

```
zaza4 <5> grep -c son results5.txt
4
```

Test

The following two questions assume that you have saved the file list3.txt and have it easily accessible. In order to answer the questions, you have to process the file using the commands we have discussed in this step.

If you sort the list on last name (ascending, the normal way), which year is the second person in the sorted list born?

The file list3.txt includes not only names and birth dates, but also ten columns with results. The next question is about these results.

What is the last name of the person with the lowest score in the third column of results (two possible answers)?

The following question assume that you have the file matrix.txt available. Answer the question by processing the file, using the commands we have discussed in this step. The file consists of 50 rows of 70 random letters each. By column N we mean the N:th letter of the row.

Cut out letters 42, 53 and 64 from each row and sort these! What sequence of three letters will come first?