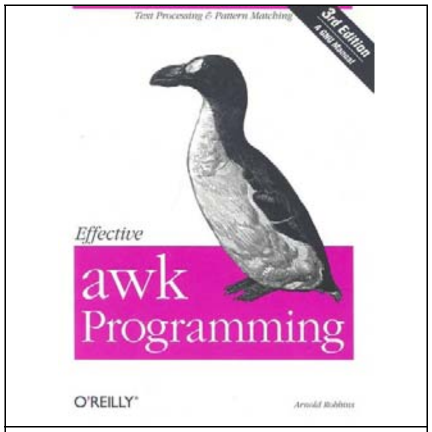
L02 – Awk, Cut, Paste, and Join

1. Awk

Awk will be your best friend. If you know how to use awk you can just throw Excel in the trash and ponder why anyone ever decided it was a good idea to write Excel in the first place. So, now that you know how I feel, what is awk?

Awk is a programming language. However, in the geosciences it is typically used on the command line to process text-based data. The name awk, comes from its authors names: Alfred **A**ho, Peter **W**einberger, and Brian **K**ernighan.



The O'Reilly book features an auk on the cover.

This lecture is aimed at giving you a basic working knowledge of awk. This document should just be viewed as an awk primer, for more info on all the things you can do with awk there are a ton of amazing resources available on the web. To get started let's create a simple example file to play around with. Using your favorite text editor create the following file named: **example.txt**.

File: example.txt

1	shear	5	20.00
2	compressional	10	2.00
3	anisotropy	30	3.50
4	perovskite	2	45.50
5	olivine	25	33.19

Note: in awk we refer to each line in the file as a **record**, and each column as a **field**. So, in the above example file we have 5 total records and 4 fields. Awk works by scanning through each line of text (or record) in the file and carrying out any instructions you tell it on that line.

In awk we access fields using syntax like: **$1** or **$2**. **$1** indicates that you are referring to the first field or first column.

Example 1 - Printing fields:
What is the output for the following examples?

```
>> awk '{print $2}' example.txt

>> awk '{print $1, $4}' example.txt

>> awk '{print $4, $2}' example.txt

>> awk '{print $1$2}' example.txt

>> awk '{print $0}' example.txt

>> awk '{print $1$2"-->$"$4}' example.txt
```

We can also do some simple arithmetic with awk.

Example 2 – Simple arithmetic on fields

```
>> awk '{print ($1*$3)}' example.txt

>> awk '{print ($4 - $3), ($1 + $1)}' example.txt

>> awk '{print ($3/$1), $2, (2*3.14*$1)}' example.txt

>> awk '{print int($4)}' example.txt
```

The last example shows that in addition to the simple arithmetic commands, awk also has some useful numeric functions, such as **sin**, **cos**, **sqrt**, etc. To see the full list check out the awk man page.

A real useful ability is to be able to search within the files. First, let's introduce some of the variables that are built into awk:

awk Variable name	What it stands for
FILENAME	Name of current input file
RS	Input record separator (Default is new line)
OFS	Output field separator string (Blank is default)
ORS	Output record separator string (Default is new line)
NF	Number of fields in input record
NR	Number of input record
OFMT	Output format of number
FS	Field separator character (Blank & tab is default)

These may not all make sense right now, but we'll come back to some of them later.

Example 3 – Simple sorting routines

Try these examples on for size:

```
>> awk 'NR > 3 {print $0}' example.txt

>> awk 'NR <= 3 {print $2}' example.txt

>> awk '$3 >= 10 {print $0}' example.txt

>> awk '$2 ~ /perov/ {print $0}' example.txt

>> awk '$2 !~ /perov/ {print $0}' example.txt
```

The comparison operators that awk allows are:

<	Less than.
< =	Less than or equal.
= =	Equal.
! =	Not equal.
> =	Greater than or equal.
>	Greater than.
~	Contains (for strings)
! ~	Does not contain (strings)

To make things even more interesting we can add some logic to our conditionals! In the following examples **&&** is the **AND** operator and **||** is the **OR** operator.

Example 4 – sorting with logic

```
>> awk 'NR > 2 && NR < 5 {print $0}' example.txt

>> awk '$3 > 10 && $4 > 2.5 {print $0}' example.txt

>> awk '$2 ~ /aniso/ || $2 ~ /oliv/ {print $0}' example.txt

>> awk 'NR >= 2 && $2 ~ /aniso/ || $2 ~ /oliv/ {print $0}' example.txt
```

You can also specify that awk does something either before starting to scan through the file (**BEGIN**) or after awk has finished scanning through the file (**END**).

Example 5 – BEGIN and END

```
>> awk 'END {print $0}' example.txt

>> awk 'END {print NR}' example.txt

>> awk 'END {print NF}' example.txt

>> awk 'BEGIN {print NF}' example.txt

>> awk 'BEGIN { OFS = "_"} {print $1, $2}' example.txt

>> awk 'BEGIN { FS = "o"} {print $1, $2}' example.txt

>> awk 'BEGIN {print "Example #5"} {print $2} END {print "End of Example"}'
example.txt
```

You can also set variables in awk and do operations with them. Occasionally it comes in handy.

Example 6 – awk variables

Here's a quick example that sets a variable x = 1 at the beginning and increments the variable by one at each record, printing the variable out as a new field for each record.

```
>> awk 'BEGIN {x=1} {print $0, x++}' example.txt
```

This is a slight variation on the above example.

```
>> awk 'BEGIN {x=0} {print $0,x+=10}' example.txt
```

The following table might help to make the above examples a little more transparent.

Assignment operator	Use for	Example	Equivalent to

+=	Assign the result of addition	a += 10
		d += c
			a = a + 10
			a = a + c

-=	Assign the result of subtraction	a -= 10
		d -= c
			a = a - 10
			a = a - c

*=	Assign the result of multiplication	a *= 10
		d *= c
			a = a * 10
			a = a * c

%=	Assign the result of modulo	a %= 10
		d %= c
			a = a % 10
			a = a % c

In example #3, we showed an example of using awk with a conditional.

```
>> awk 'NR > 3 {print $0}' example.txt
```

Essentially, this example states:

If the record number is greater than 3 then print out the entire line of the file. Awk also supports a syntax with **if** statements. E.g.,

```
>> awk '{if (NR > 3) print $0}' example.txt
```

is another way of doing the same thing. However, it is sometimes very useful to also have an **else** or **else if** statement to play around with. The next couple of examples show how to do this.

Example 7 – Control structures

```
>> awk '{if ($1 > 2) print $0;
else print $1}' example.txt
```

```
>> awk '{if ($1 > 2) print $0;
else if ($1 > 1) print $2;
else print $1}' example.txt
```

Using the command **printf** it is possible to format the output from awk. **Printf** is essentially the same as that in C. You define the width of the column, whether to left or right justify and the type of information that will be outputted—such as a string, floating point, or decimal number.

Example 8 – Formatted Output

```
>> awk ‘{print $1, $2, $3, $4}’ example.txt

>> awk ‘{printf( “%4d %-20s %-5d %-7.2f\n”, $1, $2, $3, $4)}’ example.txt
```

2. Cut, Paste, and Join

This section describes three utilities that are often used in conjunction with awk for quickly manipulating fields in files.

Paste

Sometimes you may want to extract columns of information from different files and combine them into one file. **Paste** is the perfect utility for this.

Consider the two files:

A.txt	**B.txt**
a1	b1
a2	b2
a3	b3
a4	b4
a5	b5

We can combine them as follows:

```
>> paste A.txt B.txt > C.txt
```

Join

If two separate files share a common field they can combined with join. Consider two files:

A.txt		**B.txt**	
Vs	7.2	Vs	6.3
Vp	11.3	Vp	12.4
Rho	6.6	Rho	5.9

Now try:

```
>> join A.txt B.txt > C.txt
```

Cut

Cut is incredibly useful for chopping up files into fields. Use the **–d** flag to specify a new delimiter, and the **–f** flag to state which fields to print out.

Consider a file as follows (**A.txt**) that uses underscores to separate fields:

Vs_7.2
Vp_11.3
Rho_6.6

One could just extract the numeric values by:

```
>>  cut –d_ -f2 A.txt
```

Another place I find cut useful for is in extracting information out of file names. For example, suppose I have a bunch of SAC files (seismograms) that look as follows:

```
>> ls

>> HRU.UU.EHZ      NOQ.UU.HHZ         GMU.UU.EHZ          CTU.UU.EHZ
```

The filename convention here looks like: **station_name.network.component**

If I want to make a list of just the station names I could do something like:

```
>> ls *UU* | cut –d. –f1 > stationlist.txt
```

3. Homework

1) Consider two files given below that each contain a set of Cartesian coordinates. Write an awk script to compute the distance between these pairs of points. Feel free to use any of the other commands we learned in this lecture as well.

x1	y1
0.0	0.0
0.5	0.1
0.75	0.2
1.0	0.3

x2	y2
0.0	0.0
-0.25	0.1
-0.5	0.2
-1.0	0.3

2) Below is a table of S-wave velocities at the coordinates given by the indicated latitude, and longitude (φ) in degrees. Create a file exactly as shown below, and write an awk command that will convert the longitudes given in the file below from the interval: $-180° \leq \varphi \leq 180°$ to the interval: $0° \leq \varphi \leq 360°$. Note: longitudes from 0° to 180° in the original file should not change. Format your output, such that you have three distinct labeled columns and add a single decimal place to both the latitude and longitude values.

Lon	Lat	dVs
-180	-10	2.3
-135	-10	2.4
-90	-10	2.0
-45	-10	1.8
0	-10	0.0
45	-10	-0.3
90	-10	-1.2
135	-10	-1.5
180	-10	0.0
-180	10	2.4
-135	10	2.6
-90	10	2.1
-45	10	1.6
0	10	-0.1
45	10	-0.4
90	10	-1.0
135	10	-1.0
180	10	0.3

3) Consider a file that looks as follows:

Vs
Vp
Rho
Vs
Vp
Rho
Vs

write an awk command that will print the total number of lines that contain the string **Vs**.

4) I have a group of SAC files named as follows:

>> HRU.UU.EHZ NOQ.UU.HHZ GMU.UU.EHZ CTU.UU.EHZ

Using awk, how can we change the names of all of these files so that the EHZ or HHZ is replaced by just Z. So, for example the first file is renamed as: **HRU.UU.Z**

5) Write an awk command that will print the sum and average of column #1 of a file. The output should look like:

>> Sum is: X; Average is: X

awk cheat sheet

# get total number of records in a file

```
awk ‘END {print NR}’
```

# If NR is equal to shell variable ‘n’ print line

```
awk ‘NR == ‘$n’ {print $0}’
```

# Sum the values along a column (column #2 in this example)

```
awk ‘{ sum += $2} END {print sum}’
```

# Print the sums of the fields of every line

```
awk '{s=0; for (i=1; i<=NF; i++) s=s+$i; print s}'
```

# Print out file with double spacing

```
awk ‘{print ; print “ “}’
```

# Print fields in reverse order

```
awk '{ for (i = NF; i > 0; --i) print $i }'
```

# if else syntax

```
awk ‘{if ($1 > 2) print $0;
else print $1}’ file
```

# Concatenate every 5 lines of input, using a comma separator between fields

```
awk 'ORS=NR%5?",":"\n"' file
```