**awk**

For information about awk, see **http://www.vectorsite.net/tsawk.html**

Awk is one of the standard GNU utilities and is part of Linux. It is named after the names of its authors: Aho, Weinberger, & Kernighan. The Awk text-processing language is useful for such tasks as:

* Tallying information from text files and creating reports from the results.
* Adding additional functions to text editors like "vi".
* Translating files from one format to another.
* Creating small databases.
* Performing mathematical operations on files that include numeric data.

Awk divides a file into records (usually lines in a text file) and each record into fields (usually text separated by whitespace). You can use awk to search and manipulate records and fields. To see how awk works, let's consider a data file as follows, saved as "coins.txt":

**gold 1 1986 USA American Eagle**

**gold 1 1908 Austria-Hungary Franz Josef 100 Korona**

**silver 10 1981 USA ingot**

**gold 1 1984 Switzerland ingot**

**gold 1 1979 RSA Krugerrand**

**gold 0.5 1981 RSA Krugerrand**

**gold 0.1 1986 PRC Panda**

**silver 1 1986 USA Liberty dollar**

**gold 0.25 1986 USA Liberty 5-dollar piece**

**silver 0.5 1986 USA Liberty 50-cent piece**

**silver 1 1987 USA Constitution dollar**

**gold 0.25 1987 USA Constitution 5-dollar piece**

**gold 1 1988 Canada Maple Leaf**

You can now invoke awk in a simple form with a search pattern (enclosed in slashes) and one or more actions (enclosed in curly brackets) if the search applies:

**awk <search pattern> {<program actions>}**

**Example: awk '/gold/ {print $0}' coins.txt**

which will search all records for 'gold' and if found print the entire record $0. Note that this could have been accomplished using 'grep' just as well. But with awk we could have printed specific fields only using $1, $2, ….

If no search pattern is specified, the action will apply to each record, but we could use (optional) conditional statements, i.e. an "if" statement. What would this line do, for example:

**awk '{if ($3 < 1980) print $3, " ",$5,$6,$7,$8}' coins.txt**

The general form of an awk invocation is:

**awk 'BEGIN {<initializations>}**

**<search pattern 1> {<program actions>}**

**<search pattern 2> {<program actions>}**

**...**

**END {<final actions>}'**

If you do not use specific initializations, variables are initialized to 0 automatically and you do not need the BEGIN pattern. Suppose we want to find out how many gold and how many silver coins we have. We could use:

**awk 'BEGIN {ng=0;ns=0}   
 /gold/ {ng++}   
 /silver/ {ns++}   
 END {print ng, ns}' coins.txt**

This would be better entered as an "awk" program, i.e. a simple text file containing the lines:

**# Count the number of gold and silver coins**

**BEGIN {ng=0;ns=0}**

**/gold/ {ng++}**

**/silver/ {ns++}**

**END {print "Gold coins: ", ng,", Silver coins: ", ns}**

You can invoke this program using awk with the –f switch, as in:

**awk -f count\_coins.awk coins.txt**

Multiple programming statements can be separated by semicolons.

You can also use the –F switch to change the separation character. For example, if fields are separated by colons, use "awk –F: '/search/ file.txt"

**Exercise:** Print all usernames (they are stored in /etc/passwd) in alphabetical order, but only the usernames.

**Exercise**: Find the current price of gold and silver on the web. Then write an awk program that computes the value of your coin collection.

**Web Use:** How to use this in our Address book program? We could add more fields to our database, such as email, street address, city, country, and then use awk to selectively search different fields only instead of using grep for a search across all records.

**Special variables and Loops**

Awk knows a number of special variables. The most useful one is NF for the number of fields in each record, while NR contains a count of the current line. For example:

**awk '{ print $0 " has " NF " words" }' sample.txt**

You can also use loops by using the for construct:

**awk 'BEGIN {for (i=10; i<=20; i+=2) print i}' sample.txt**

You can also use "for (var in array)" to iterate through arrays, but in an unpredictable order.

**Arrays in awk**

Awk also permits the use of arrays. The naming convention is the same as it is for variables, and, as with variables, the array does not have to be declared. Awk arrays can only have one dimension; the first index is 1. Array elements are identified by an index, contained in square brackets. For example:

**some\_array[1], some\_array[2], some\_array[3] ...**

One interesting feature of Awk arrays is that the indexes can also be strings, which allows them to be used as a sort of "associative memory". For example, an array could be used to tally the money owed by a set of debtors, as follows:

**debts["Kim"], debts["Roberto"], debts["Vic"] ...**

**Exercise**: Use a "for" loop and an associative array to count the number of words in a text file.

Hint on for loops. The variable NF stands for the "number of fields" in a line. So, for example:

**for (i=1; i<=NF; i++)  
 print $i**

would print all fields in a line. About for loops, you can use "in" with arrays to iterate through an array. So, for example:

**BEGIN {a["Bert"]=1; a["Carl"]=2; a["Jane"]=3;  
 for (word in a)   
 print "Word = " word " and a[word] = " a[word] }**

Also, awk includes a build-in function "tolower" that you can use to remove the distinctions in case. Finally, you can use 'sed' to filter out periods, commas, etc., and "sort" to sort the output.

**Exercise:** Write a program to show the frequencies of three particular words in a text file.

**Exercise (web use)**: Write a program to print out the frequencies of URL's in your web access log.

For detailed information about awk, see **http://www.vectorsite.net/tsawk.html**