AWK QUICK REFERENCE

From An Awk Primer:
http://www.vectorsite.net/tsawk.html

* This final section provides a convenient lookup reference for Awk programming. If you want a more detailed reference and are using a UN*X or Linux system, you might look at the online awk manual pages by invoking:

```
man awk
```

Apparently some systems have an "info" command that is the same as "man" and which is used in the same way.

* Invoking Awk:

```
awk [-F<ch>] {pgm} | {-f <pgm file>} [<vars>] [-|<data file>]
```

-- where:

```
ch:            Field-separator character.
pgm:           Awk command-line program.
pgm file:      File containing an Awk program.
vars:          Awk variable initializations.
data file:     Input data file.
```

* General form of Awk program:

```
BEGIN                 {<initializations>}
 <search pattern 1>  {<program actions>}
 <search pattern 2>  {<program actions>}
...
END                   {<final actions>}
```

* Search patterns:

```
/<string>/       Search for string.
/^<string>/      Search for string at beginning of line.
/<string>$/      Search for string at end of line.
```

The search can be constrained to particular fields:

```
$<field> ~ /<string>/     Search for string in specified field.
$<field> !~ /<string>/    Search for string \Inot\i in specified field.
```

Strings can be ORed in a search:

```
/(<string1>)|(<string2>)/
```

The search can be for an entire range of lines, bounded by two strings:

```
/<string1>/,/<string2>/
```

The search can be for any condition, such as line number, and can use the following comparison operators:

```
== != < > <= >=
```

Different conditions can be ORed with "||" or ANDed with "&&".

```
[<charlist or range>]    Match on any character in list or range.
[^<charlist or range>]   Match on any character not in list or range.
.                          Match any single character.
*                           Match 0 or more occurrences of preceding string.
?                           Match 0 or 1 occurrences of preceding string.
+                           Match 1 or more occurrences of preceding string.
```

If a metacharacter is part of the search string, it can be "escaped" by preceding it with a "\".

* Special characters:

```
\n      Newline (line feed).
```

Backspace. \r Carriage return. \f Form feed. A "\" can be embedded in a string by entering it twice: "\\".

* Built-in variables:

```
$0; $1,$2,$3,...   Field variables.
```

```
   NR                   Number of records (lines).
   NF                   Number of fields.
   FILENAME             Current input filename.
   FS                    Field separator character (default: " ").
   RS                    Record separator character (default: "\n").
   OFS                   Output field separator (default: " ").
   ORS                   Output record separator (default: "\n").
   OFMT                  Output format (default: "%.6g").
```

* Arithmetic operations:

```
   +    Addition.
   -    Subtraction.
   *    Multiplication.
   /    Division.
   %    Mod.
   ++   Increment.
   --   Decrement.
```

Shorthand assignments:

```
   x += 2   -- is the same as:   x = x + 2
   x -= 2   -- is the same as:   x = x - 2
   x *= 2   -- is the same as:   x = x * 2
   x /= 2   -- is the same as:   x = x / 2
   x %= 2   -- is the same as:   x = x % 2
```

* The only unique string operation is concatenation, which is performed simply by listing two strings connected by a blank space.

* Arithmetic functions:

```
   sqrt()       Square root.
   log()        Base \Ie\i log.
   exp()        Power of \Ie\i.
   int()         Integer part of argument.
```

* String functions:

- length()

Length of string.

- substr(<string>,<start of substring>,<max length of substring>)

Get substring.

- split(<string>,<array>,[<field separator>])

Split string into array, with initial array index being 1.

- index(<target string>,<search string>)

Find index of search string in target string.

- sprintf()

Perform formatted print into string.

* Control structures:

```
    if (<condition>) <action 1> [else <action 2>]
    while (<condition>) <action>
     for (<initial action>;<condition>;<end-of-loop action>) <action>
```

Scanning through an associative array with "for":

```
   for (<variable> in <array>) <action>
```

Unconditional control statements:

```
   break         Break out of "while" or "for" loop.
   continue      Perform next iteration of "while" or "for" loop.
   next          Get and scan next line of input.
   exit           Finish reading input and perform END statements.
```

* Print:
   print <i1>, <i2>, ... Print items separated by OFS; end with newline.
   print <i1> <i2> ... Print items concatenated; end with newline.

* Printf():

General format:
    printf(<string with format codes>,[<parameters>])

Newlines must be explicitly specified with a "\n".

General form of format code:
    %[<number>]<format code>

The optional "number" can consist of:

- A leading "-" for left-justified output.

- An integer part that specifies the minimum output width. (A leading "0" causes the output to be padded with zeroes.)

- A fractional part that specifies either the maximum number of characters to be printed (for a string), or the number of digits to be printed to the right of the decimal point (for floating-point formats).

The format codes are:

d	**Prints a number in decimal format.**
o	**Prints a number in octal format.**
x	**Prints a number in hexadecimal format.**
c	**Prints a character, given its numeric code.**
s	**Prints a string.**
e	**Prints a number in exponential format.**
f	**Prints a number in floating-point format.**
g	**Prints a number in exponential or floating-point format.**

* Awk can perform output redirection (using ">" and ">>") and piping (using "|") from both "print" and "printf".