GAWK

The University of Arizona
Computer Science Department
College of Science
CSC 520, Spring 2008
Principles of Programming Languages
Dr. Christian Collberg

Team Members:
Amit Wadhwa
Nithya Chandrasekaran

May 14, 2008

Contents

1. Introduction

GAWK is a highly powerful text processing and pattern matching language that enables working with information either stored in files or piped to it as input. GAWK follows the data driven paradigm such that each program consists of a set of rules applicable to a pattern or a piece of content. The programs are interpreted which means that is they are converted to binary on the fly each time they run, by an interpreter program. This renders them portable.

When a pattern occurs repetitively in a text file or a similar action needs to be performed for multiple match patters, single-use programs in C, C++, or Pascal serving the purpose is time-consuming and inconvenient. GAWK handles such tasks very easily. The gawk utility interprets a special-purpose programming language that makes it easy to handle simple data-reformatting jobs. The gawk language is also very useful for processing large amounts of raw data from the output of other utility programs like 'ls'. Programs in gawk are usually much smaller than they would be in other languages serving the same purpose.

2. History

AWK derived its name from its designers. Alfred V. Aho, Peter J. Weinberger and Brian W. Kernighan. The first version of AWK came into being in 1977 at AT&T Bell Laboratories. An enhanced with better support for user defined functions, multiple streams and computed regular expressions was introduced in 1985. The GNU implementation of GAWK was written by Paul Rubin in 1986. It was later completed by Jay Fenlason. Jürgen Kahrs introduced the network access features in GAWK.

The System V Release version 4 cleaned up some more behaviors of the language and clarified the specification for awk in the POSIX command Language and Utilities standard. Some new features added between V7 and SVR3.1 included support for built in numeric functions, string manipulation functions, C compatible operator precedence.

3. Uses

Some applications of GAWK include:

1. Enables select/display and other operations on contents of the file, selecting rows, columns or fields and sorting as necessary (Text Processing).
2. Manage small, personal databases.
3. Generate formatted reports.
4. Prototyping.
5. Validate data.
6. Generate indexes and perform other document preparation.
7. Perform calculations with numeric information extracted from a specific column.

The broader application areas on which the focus of GAWK lies are:

1. Computer Networking
2. Artificial Intelligence
3. Data Mining
4. Bio Informatics

4. Generic Gawk Command

The generic command in GAWK is a *pattern-action* statement and a gawk program usually consists of one or more of these statements

- /pattern/{ action}

Gawk scans a sequence of input lines one after another, searching for lines which match the patterns in the program. For each matching pattern the corresponding action is performed.

5. Patterns in GAWK

Actions are performed when a line matches the associated pattern. One of the powerful aspects of Gawk is that the user can specify patterns in various ways. The various patterns are:

1. BEGIN and END
2. Expression based patterns
3. Regular expression patterns
4. Compound patterns
5. Range patterns

5.1 BEGIN and END

Gawk contains two special patterns *BEGIN* and *END.* These patterns do not match any input line but are used for initialization and cleanup respectively. The statements in BEGIN action are executed before gawk reads any input and those in the END action are executed after all input has been read.

5.2 Expression Patterns

Any expression can be used a pattern. If the expression evaluates to a nonzero or nonnull value at an input line, then the pattern matches that line. Typical expressions would involve comparison between strings or numbers using any of the comparison operators (<, <=,>, >=,!=,==,~,!~), where ! stands for not (!= : not equal to) and ~ stands for matched by.

Example:

$3>=$2	Select lines where value of third field is greater than or equal to second field
$0>”M”	Select lines which start with ‘N’, ‘O’ …

5.3 Regex Patterns

In gawk, regular expressions patterns can be used for specifying and matching strings.

Example:

/AB*C/	Matches AC, ABC, ABBC etc
/[A-Z]+/	Matches any string of one or more uppercase characters
/^ [0-9]+$/	Matches any line that consists of only digits
/[A-Za-z][0-9]?$/	A letter or a letter followed by a digit

5.4 Compound Patterns

A compound pattern can be used to combine multiple patterns using parentheses and logical operators OR (||), AND (&&) and ! (NOT). A compound pattern matches an input line if the expression evaluates to true.

The following expression uses the AND operator to select all lines in which the fourth field is 'Readme.txt' and third field exceeds 500.

$4=="Readme.txt" && $3>500

5.5 Range Pattern

A range pattern consists of two patterns separated by a comma, i.e. pat1, pat2

It matches each line between an occurrence of pat1 and the next occurrence of pat2. E.g. /hello/, /world/ matches lines starting with the first line that contains 'hello' till the next line containing 'world'.

6. Actions in GAWK

In the pattern-action statement the pattern decides when the action has to be executed. The action can sometimes be very simple like a print statement and other times it can be a sequence of statements which are separated by a newline or a semicolon in GAWK.

Actions can consist of:

1. Expressions (assignments using variables/arrays, some operations)
2. Function calls including Print, printf statements
3. Control Statements: Conditionals(if and if else), Loops(while, for, for in, do) and Other flow control statements(break, continue, next, exit, exit expression etc)

6.1 Variables and Arrays in GAWK

A variable in Gawk is very similar to variables in C with a few differences. Firstly, the type of a variable in Gawk is not declared, instead Gawk infers the type from the context. An uninitialized variable has the string value "" (null string) and the numeric value of 0.

Gawk also has some built in variables called predefined variables that exist to provide easy access to useful information and to change the default behavior of Gawk.

Some of the variables are:

Variable	**Meaning**
ARGC	The number of command-line arguments
ARGV	An array of command-line arguments
ERRNO	The UNIX system error message
FS	The input field separator
IGNORECASE	Controls the case sensitivity
NF	The number of fields in the current record

An array in Gawk is a table of values known as elements. Each element has a unique index which can be a string or a number. Gawk basically supports only 1 dimensional array, but multi dimensional arrays can be simulated as we will see in a later part of the report. Gawk arrays are

quite different from arrays in other languages since the size of an array needn't be specified you start to use it and any number or even a string may be used as an array index.

In most languages, array is stored in a contiguous block of memory and the index in the array must be a positive number with index 0 specifying the first element, index 1 the second element and so on.

Index: 0 1 2

4	5	9

In Gawk arrays are associative i.e. each array is a collection of pairs, an index, and its corresponding array element value

Element	Value
2	5
"foo"	8
3	"bar"

One advantage of such a scheme is that pairs can be added to the array anytime. Another result of associative arrays is that we can have non-positive numbers and strings as indices.

Example: Using Arrays

```
{ line[NR] ] = $0 }
  END {  i = NR;
  while( i > 0){
          print line[i];
          i=i-1;
           }
}
```

Apart from the above Gawk offers a special *in* loop which can be used to iterate through array elements.

E.g. for (x in myarray) { print myarray[x] }

Multi dimensional arrays can be simulated with GAWK in the following way:

E.g. myarray[1,3] = "a"; myarray[1,"ppl"] = 5;

Internally, Gawk stores the index as subscript1@subscript2 i.e.

myarray[1,3] = "a"; is equivalent to *myarray[1@3]="a".*

This property can be used to implement data structures like lists and set easily with arrays. E.g. mylist[1,"value"]=2; mylist[1,"next"] = 3;

6.2 Functions

GAWK has a lot of built in functions and supports user defined functions also. *Built-in* functions are functions that are always available for the GAWK program to call. The major Built in functions are for manipulating numbers and strings and for Input/output.

User defined functions have to be defined anywhere between the rules of the GAWK program. Thus, the general form of a GAWK program is extended to include sequences of rules *and* user-defined function definitions.

The definition of a function looks like this:

function *name* (*parameter-list*) {
    body-of-function
}
The keyword function may be abbreviated func.

6.3 Control Statements

Gawk offers various control statements like if-else, while to control the flow of execution of the program. Most of the control statements in Gawk are patterned on similar statements in C. The various control statements are:

1. If
2. While
3. do-while
4. for
5. break
6. continue
7. next
8. exit

All other statements apart from *next* and *exit* are same as C. The next statement forces Gawk to immediately stop processing the current record and go on to the next record. The exit statement causes Gawk to immediately stop executing the current rule and to stop processing input; any remaining input is ignored. To see the details of other control statements please see references [1], [2], or [4].

7. Basic GAWK Programs

7.1 One Liners

GAWK supports simple yet powerful command line tools. Some command line tools are listed below:

Example 1:

gawk '{print $1}' file.data
reads the file ‘file.data’ and performs the action of printing the first field of every record.

Example 2:

ps |gawk '$11 == "ssh" {print $2}’

This command pipes the output of unix ‘ps’ command and displays those pid’s whose filename matches ‘ssh’.

Example 3:
Regular expression pattern identification can also be done using command line expression.

gawk '$2!~/pattern1/{ print $9,$1 }' filename | sort

Example 4:
Various conditionals can be imposed to extract required fields from only those records which satisfy a predicate.

gawk '$3 > $2{print $6}' filename

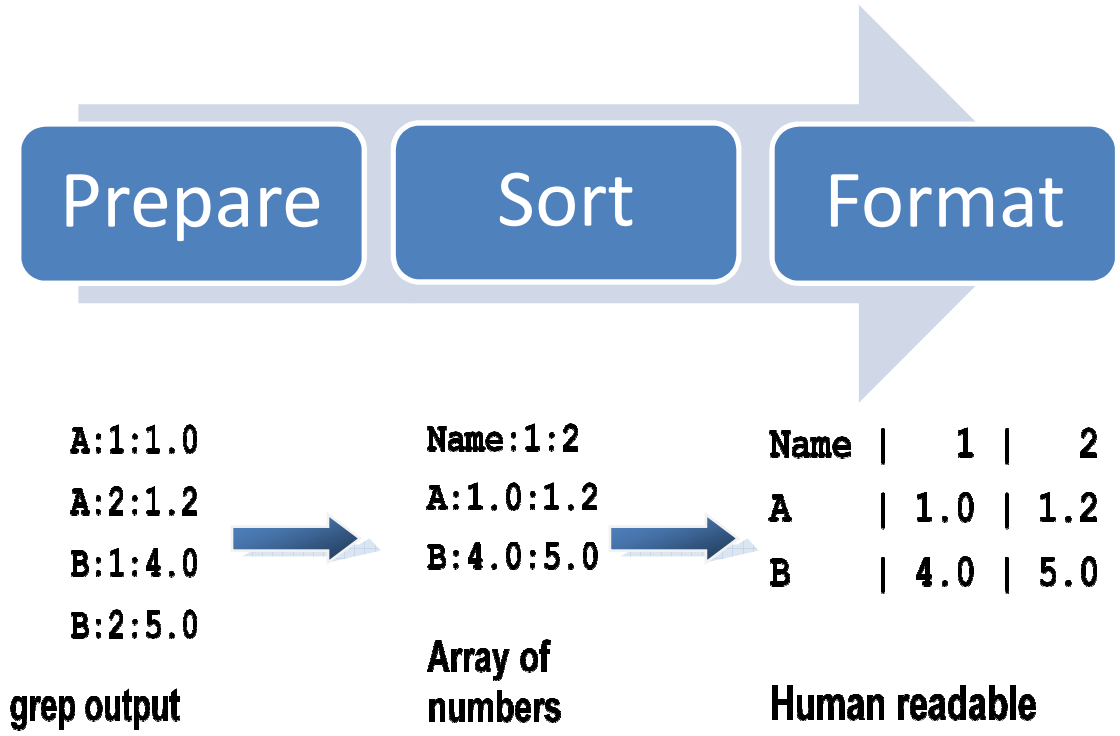
This example prints the 6th field only in those records wherein the third field is greater than the second field value.

7.2 GAWK Scripts

Every line in a gawk program is addressed as a record and is composed of various fields, each field being an independent entity on which an action can be applied.

File = records + field

The input/output streams are ASCII text records. They may be treated as numeric quantities, strings, characters or fields.
A Gawk script has three parts:



Example: The following illustrates a gawk program with well defined BEGIN and END blocks
BEGIN{print "start"}
$1=="start"{print "process"}
END{print "end"}

GAWK is also used in pretty formatting and display of reports. A variety of operations can be used to display the output. Sorting, summation of two columns and conditional display are just a few listed among a plethora of auxiliary actions supported by gawk.HTML tags can be included as a part of the script to enhance the appearance and readability of the report and to build other useful tools like recursive directory index creation.

In gawk, the output of a command can also be piped into getline, using `*command* | getline'.The string 'command' is run as a shell command and its output is piped into gawk to be used as input. This form of input using 'getline' reads one record at a time from the pipe

Example: The following is a code snippet extracted from a program that builds an html index file for the current directory and all its sub directories recursively. The second part of the program illustrates how HTML tags can be used as a part of gawk program to build index pages in a formatted manner.

```
#!/usr/bin/gawk -f
BEGIN {
FS= " ";
  listfilecomm = "ls -lh " ARGV[1];
   while( listfilecomm  | getline )
    {

          if(flag == 0)
          {
          total=$NF;
          flag=1;
           continue;
           }

           system ("gawk -f sample.gawk "result"/"$NF " > "result"/"$NF"/"$NF".html");
           <-- do other stuff to keep track of the files read --->
           printf "   <TR>\n"
                    printf "      <TD align=left colspan=500>
                              <a ref=\"%s\">%s</a></TD>\n",href, $NF;
                    printf "      <TD align=left colspan=20 >%s</TD>\n", size;
                    printf "      <TD align=left colspan=20>%s %s</TD>\n", $6, $7;
                    printf "      <TD align=left colspan=20>%s</TD>\n", $8;
                    printf "      <TD align=left colspan=20>%s</TD>\n", type;
           printf "   </TR>\n"

}
```

(This program is not in its entirety and only specific portions have been extracted for illustration)

8. RUNNING GAWK

There are two ways to run gawk:

1. Running from a script
2. Running from a Command line

Brief details of both the methods are given in this section.

8.1 Running from script

By invoking the command,

gawk 'program-name' input-file1 input-file1

the shell starts gawk and uses the program 'program-name' to process the records in the input file. The program, 'program-name' contains a series of rules (pattern/action) to be applied to the input files that follow it. If the gawk program name is mentioned without supplying an input file name, then whatever is typed next is taken as input for the gawk program.

Example:

A sample gawk program is given as follows

```
#! /bin/gawk -f
BEGIN    { print "I love GAWK" }
```

If the script file name is gawky, then it can be directly invoked by calling

$gawky (on the command line)

This will have the same effect as

$gawk –f gawky (-f option fetches the program to be executed)

8.2 From the command line

By invoking the /pattern/{action} and the file name on which it is to be applied as follows.

cat file | gawk '(pattern){action}'

9. Concepts

In this section we will see how GAWK relates to the concepts we have discussed in the class. One may notice most of the principles guiding the language are based on the assumption that GAWK will normally be used to write small programs (max. few hundred lines).

1. Interpreted: GAWK is interpreted which seems suitable to its primary usage of writing small programs which are around one to a few hundred lines. However, the latest gawk version comes with a compiler.
2. No predefined memory limit, so the size of memory allocated or the files being processed are limited only by the memory made available by the OS (which may be dependant of amount of primary memory, virtual memory etc)
3. No explicit type system, variable types are interpreted by their usage.
4. In control structures supports user defined functions. Has an *in* statement which acts like an Array iterate.
5. In Error Handling it doesn't support exception handling but unlike AWK, GAWK gives meaningful error messages.

Some unique features in GAWK which make it quite different from other mainstream languages are:

1. Associative Arrays (Refer Section 6.1)
2. Automatic Initialization and Implicit Coercion of variables (Refer Section 6.1)
3. Lint Checking when using the compiler which warns about potential runtime bugs
4. Support for TCP/IP communication:
   GAWK doesn't use sockets like C, C++, or Perl for network communication. Gawk extends the two-way I/O mechanism to simple networking through the use of special file names and the operator |&. When a "co process" that matches the special files we are about to describe is started, gawk creates the appropriate network connection, and then two-way I/O proceeds as usual.

   Example: Ask the machine about the current time

   BEGIN{"/inet/tcp/0/localhost/daytime" |& getline
   print $0 close("/inet/tcp/0/localhost/daytime")}

10. GAWK Internals

Gawk has transparent memory management built into it meaning it just needs to be told what is to be done and it handles much of the hard work behind the scenes.

When a text file is fed as an input to GAWK, the following things actions are taken internally.

1 All the files listed in the command line are opened and read automatically.
2 Memory management for all variables is handled.
3 Each line is parsed and each line is split into fields based on the field separator.
4 Each record/line in the file is presented to the script as $0.
5 The required internal variables (NF, NR, FS, and RS) are automatically populated during the parsing action.
6 Conversion between the internal data types (String, arrays, and numbers) is automatically handled.
7 Once the processing is done, the text file is closed.

New features maybe added to GAWK in the form of extension modules with the support of shared libraries. The gawk coding style needs to be followed for GNU compliance. As a part of a brief description of Gawk internals, some structure members, functions, and macros which are used when writing extensions are listed.

1 NODE: Strings, variables, numbers as well as arrays and all other types are built using objects of type Node.
2 AWKNUM: This represents the internal type of floating point number.
3 n->param_cnt: The number of parameters passed to the function at runtime.
4 n->stlen: The data and length of a node's string value.
5 n->type: This value should either be Node_var or Node_var_array for function parameters.
6 NODE *make_string(char *s, size_t len) : This takes a C string and turns it into a pointer to a NODE that can be stored appropriately in permanent storage.
7 NODE *make_number(AWKNUM val): This takes an AWKNUM and turns it into a pointer to a NODE that can be stored appropriately in permanent storage.

Dynamic extension and addition of new built-in functions to gawk is possible via 'extension(lib,func)' which dynamically loads the shared library and calls function in it to initialize the library. The environment variable AWKPATH specifies a search path to use when finding source files named with the −f (fetch) option. The default path is set to ".:/usr/local/share/awk", if this variable is absent. No path search is performed if a file name given to the −f option contains a ''/'' character.

11. Comparison

This section addresses some similarities and differences between gawk and other some other popular scripting languages.

11.1 Perl vs. Gawk

The focus of Perl lies in much larger application development unlike gawk which is powerful for small programs. Perl is semi-compiled by the standard interpreter before execution starts, giving it a speed advantage on some programs. But gawk is entirely interpreted.

11.2 Python vs. Gawk

Gawk saves a lot of code by implicitly processing the command line, including options and arguments and reading and looping over the input as well as doing field splitting. In gawk, gsub allows one to compactly obtain the list of whitespace whereas in Python, this needs to be done by hand. In Python output in a buffer has to be built up, whereas in gawk printf does this automatically. Python lacks the command processing, option processing features and does not allow sub and gsub functions.

12. Summary

GAWK is a very simple and powerful language. Its power resides in accomplishing tasks with a few lines of code which can be intuitively understood by a reader. It has powerful text processing and pattern matching capabilities which can make tasks of reading raw data and transforming it into formatted reports.

The main drawback of GAWK is that it is mainly line oriented in nature and has no procedural or object oriented or functional concepts which makes it hard to write larger and complex applications with it.

13. References

Books

1. The AWK Programming Language by Aho, Weinberger, Kerningham.
2. Effective AWK programming by Arnold Robbins.
3. Sams Red Hat 6 unleashed

Web

4. GAWK manual http://www.cs.utah.edu/dept/old/texinfo/gawk/gawk_toc.html
5. Introduction to GAWK http://www.linuxjournal.com/article/1156
6. Getting started with GAWK https://www6.software.ibm.com/developerworks/education/au-gawk/index.html