CS29206: Systems Programming Lab

Autumn 2025

Programming with awk

Resources

- "A Practical Guide to Linux" by Mark Sobel (Chapter 14)
- "Your Unix/Linux: The Ultimate Guide" by Sumitabha Das (Chapter 12)
- Slides from Prof. Abhijit Das's page (https://cse.iitkgp.ac.in/~abhij/course/lab/SPL/Spring24/slides/gawk.pdf)
  - This set of slides we are teaching from is mostly copied from the above slides, with a different primary example and some other changes. However, the examples and the practice problems in the above slides are also very useful and you should see them.

The Unix command awk

- Named after the designers Alfred V. Aho, Peter J. Weinberger, and Brian W. Kernighan
- We discuss the GNU version gawk
- awk is a powerful programming language for pattern processing
  - Goes through a file line-by-line, and takes specified actions if specified patterns are encountered
- Run the command as

```
gawk <OPTIONS> 'COMMANDS' <FILE(S)>
```

- If the commands are written in a file named COMMANDFILE, run as

```
gawk <OPTIONS> -f COMMANDFILE <FILE(S)>
```

Records and Fields

- awk reads the input file(s) line by line
- Each line is called a record
- Each record is split into fields by a separator
- The default field separator is space or tab
- You can specify your separator by running with the -F option.

```
gawk -F: 'COMMANDS' <FILE(S)>
```

- The current record is accessed as $0
- The individual fields are accessed as $1, $2, $3, . . .

An Example File

Consider a file student.txt storing students' details (with : as a field separator). The fields are names (first, middle, last), type (UG/PG/RS), roll no., and list of subjects enrolled (separated by comma) and marks obtained in them (out of 100, separated by comma).

```
Ajay::Gupta:UG:10CS20010:Algorithms,AI,Networks:75,80,85
Abhik::Das:UG:10MA20012:Algorithms,OS,Networks,ML:70,90,80
Riya::Sinha:UG:10CS20013:OS,Networks,ML:85,70,80
Alok:Kumar:Roy:PG:10CS60014:OS,ML,AI:70,70,75
Arvind::Srinivasan:PG:10MA60012:Algorithms,OS,ML:80,70,75
Gautam::Kumar:PG:10MA60024:Algorithms,AI:70,75
Anusha:V:Pillai:RS:09MA10001:Networks,AI:75,70
Ashok:Lal:Gupta:RS:09CS10013:Algorithms,ML:80,85
```

- Consider the line

  Abhik::Das:UG:10MA20012:Algorithms,OS,Networks,ML:70,90,80
- Here : is used as the field separator
- gawk will have the strings stored in the following variables
  - $0 = “Abhik::Das:UG:10MA20012:Algorithms,OS,Networks, ML:70,90,80”
  - $1 = “Abhik”
  - $2 = “”
  - $3 = “Das”
  - $4 = “UG”
  - $5 = “10MA20012”
  - $6 = “Algorithms,OS,Networks,ML”
  - $7 = “70,90,80”

The commands

- There is an optional BEGIN section that is executed before any record is read
- This is followed by reading the records one by one, and performing actions driven by a set of patterns
- Finally, there is an optional END section that is executed after all records are read
- For each record, only those actions are taken for which the record matches the corresponding patterns
- The actions are taken in the sequence given in the program
- An empty pattern matches every record

```
BEGIN { Initial actions }
PATTERN1 { Action1 }
PATTERN2 { Action2 }
. . .
PATTERNn { Actionn }
END { Final actions }
```

printType.awk

```
BEGIN {
    FS = ":"
    print "Reading the student database ..."
}
{  print $1 "  "  $2 "  " $3 }
$4 == "UG" { print "\tType: Undegraduate" }
$4 == "PG" { print "\tType: Postgraduate"}
$4 == "RS" { print "\tType: Research Scholar"}
{ print "\tRoll Number: " $5}
{
   print "\tSubjects taken"
   n = split($6, clist, ",")
   for (i=1; i<=n; ++i) { print "\t\t" clist[i] }
}
END { print "That is all I have. Bye..." }
```

```
$ gawk -F: -f printType.awk student.txt
```

```
Reading the student database ...
Ajay    Gupta
        Type: Undegraduate
        Roll Number: 10CS20010
        Subjects taken
                Algorithms
                AI
                Networks
Abhik    Das
        Type: Undegraduate
        Roll Number: 10MA20012
        Subjects taken
                Algorithms
                OS
                Networks
                ML
Riya    Sinha
        Type: Undegraduate
        Roll Number: 10CS20013
        Subjects taken
                OS
                Networks
                ML
Alok  Kumar  Roy
        Type: Postgraduate
        Roll Number: 10CS60014
        Subjects taken
                OS
                ML
                AI
Arvind    Srinivasan
        Type: Postgraduate
        Roll Number: 10MA60012
        Subjects taken
                Algorithms
                OS
                ML
Gautam    Kumar
        Type: Postgraduate
        Roll Number: 10MA60024
        Subjects taken
                Algorithms
                AI
Anusha  V  Pillai
        Type: Research Scholar
        Roll Number: 09MA10001
        Subjects taken
                Networks
                AI
Ashok  Lal  Gupta
        Type: Research Scholar
        Roll Number: 09CS10013
        Subjects taken
                Algorithms
                ML
That is all I have. Bye...
```

- Could have written printType.awk to have a single action

```
BEGIN {
    FS = ":"
    print "Reading the student database ..."
}
{
    print $1 "  " $2 "  " $3
    if ($4 == "UG") { print "\tType: Undegraduate" }
    if ($4 == "PG") { print "\tType: Postgraduate"}
    if ($4 == "RS") { print "\tType: Research Scholar"}
    print "\tRoll Number: " $5
    print "\tSubjects taken"
    n = split($6, clist, ",")
    for (i=1; i<=n; ++i) { print "\t\t" clist[i] }
}
END { print "That is all I have. Bye..." }
```

Filtering by Pattern Matching

- The pattern can be any regular expression, enclosed with a pair of delimiters (usually /)
- Here is an example of printing out roll no.s of all UG and PG students separately who has taken OS.

```
BEGIN {
    FS = ":"
    nUG = 0;
    nPG=0;
}
{
    if ($6 ~ /OS/) {
      if ($4 ~ /UG/) { nUG++; UG_OS[nUG] = $5}
      else {nPG++; PG_OS[nPG] = $5}
    }
}
END {
     print nUG " UG students have taken OS"
     for (i=1; i<=nUG; i++) print "\t" UG_OS[i]
     print nPG " PG students have taken OS"
     for (i=1; i<=nPG; i++) print "\t" PG_OS[i]
}
```

```
$ gawk  -F:  -f OStaken.awk  student.txt
2 UG students have taken OS
        10MA20012
        10CS20013
2 PG students have taken OS
        10CS60014
        10MA60012
```

Similarities with C

- awk syntax is quite similar to C syntax
- Comparison operators: ==, !=, <, <=, >, >=
- **New operator:** ~ (pattern matching) and !~ (pattern non-matching)
- Logical operators: &&, | |, and !.
- Arithmetic operators: +, -, *, /, %, ++, --.
- **New operator:** ** (exponentiation).
- Assignment operators: =, +=, -=, *=, /=, and %=
- if and if else statements
- while and for loops, break, and continue
- printf and sprintf work exactly as in C

Built-in Variables

$0:	The current record
$1,$2,$3,. . .	The fields in the current record
RS	The Record Separator (default: newline)
NR	The Number of the current Record (1, 2, 3, . . .)
FS	Field Separator
NF	The Number of Fields in the current record
FILENAME	The name of the current file being read (NULL if the input is taken from the keyboard (stdin)

Variables and Arrays

- Variables are not needed to be declared before use
- Numeric variables are automatically initialized to 0
- String variables are automatically initialized to the empty string
- Variables do not have fixed types
- Strings and numbers are treated in a unified manner
- If a string is used in a numerical context, it is automatically converted to a number if it is a numeric string, or to 0 otherwise
- A number is automatically converted to a numeric string (like during printing)
- Strings are compared with respect to the lexicographic ordering. For example, 9 < 10 (as numbers), whereas "9" > "10" (even though both are numeric)
- Array indexing is 1-based

Some new built-in functions

int(x)	The integer part of x
length(s)	Length of the string s
index(s,t)	Index of the substring t in the string s (0 if t is not a substring of s)
substr(s,b,l)	Substring of the string s beginning at index b and of length l
toupper(s)	Copy of the string s converted to upper case
tolower(s)	Copy of the string s converted to lower case
split(s,A,d)	Split the string s with respect to the delimiter (a string again), and store the parts in the array A. The number of parts obtained by splitting s (the size of A) is returned.

Example

- Listing the subjects taken by all students of Mathematics department (MA in the roll number)
- Since BEGIN is optional, so we skip it

```
{
    dept = substr($5, 3, 2)
    if (dept == "MA" ) {
         printf("%s %s %s with roll no. %s  has taken courses %s\n",
$1, $2, $3,  $5, $6);
          nst++;
     }
}
END {
    printf("Number of Math dept. students are %d\n", nst);
}
```

Can also replace the lines

```
    dept = substr($5, 3, 2)
    if (dept == “MA” ) {
```

with

```
    pos = index($5, “MA”)
    if (pos != 0) {
```

```
$gawk -F: -f fmath.awk  student.txt
Abhik  Das with roll no. 10MA20012  has taken courses Algorithms,OS,Networks,ML
Arvind  Srinivasan with roll no. 10MA60012  has taken courses Algorithms,OS,ML
Gautam  Kumar with roll no. 10MA60024  has taken courses Algorithms,AI
Anusha V Pillai with roll no. 09MA10001  has taken courses Networks,AI
Number of Math dept. students are 4
```

Associative Arrays

- Arrays can be indexed by strings
- The syntax is the same: Array[string]
- Here, string is not automatically converted to an integer index
- **Note:** Array[5] and Array["5"] are different
- Loops can be used on associative arrays as:

```
for (name in Array) {
    # Access entries as Array[name]
}
```

- Iterations are not in the sorted order of names ctaken.awk

```
{
      if ($4 == "UG") {
         n = split($6, ctaken, ",")
         for (i=1; i<=n; i++) { courses[ctaken[i]] = 
courses[ctaken[i]] " " $5 }
      }
}
END {
   for (c in courses) { printf("%s: %s\n", c, courses[c]) }
}
```

Example: Subject-wise listing of all UG students

```
$gawk  -F:  -f ctaken.awk  student.txt
OS:  10MA20012 10CS20013
AI:  10CS20010
Algorithms:  10CS20010 10MA20012
ML:  10MA20012 10CS20013
Networks:  10CS20010 10MA20012 10CS20013
```

User-defined functions and run-time user inputs

fib.awk

```
function F ( n )
{
    if (n <= 1) { return n }
    return F(n-1) + F(n-2)
}
BEGIN {
   printf("Enter a positive integer: ")
   getline n < "-"
   n = int(n)
   print "Fib(" n ") = " F(n)
}
```

```
$gawk  -f fib.awk
Enter a positive integer: 8
Fib(8) = 21
```

Setting variable values using –v option

fib.awk

```
function F ( n )
{
        if (n <= 1) { return n }
        return F(n-1) + F(n-2)
}
BEGIN {
        printf("Entered argument: %d\n", n)
        print "Fib(" n ") = " F(n)
}
```

```
$gawk -v n=6 -f fib.awk
Entered argument: 6
Fib(6) = 8
```

Scope of variables

- All variables used are global
- There is no provision for declaring local variables
- Only the function parameters act as local variables
- Parameter passing is by value only
- If you want to use local variables in a function, do the following
  - Add your local variables to the list of parameters
  - You do not need to pass values to all the parameters
  - Any value not passed is initialized to 0 or the empty string

```
function oddsum ( n )
{
    sum = 0
    term = 1
    for (i=1; i<=n; ++i) {
        sum += term
        term += 2
    }
    return sum
}
BEGIN {
    n = 10
    sum = 0
    for (i=1; i<=n; ++i) {
        print "Calling oddsum(" i ")"
        sum += oddsum(i)
        print "sum = " sum
     }
     print sum
}
```

The output if you run it is 162

This is because n is a local variable, but i and sum are global variables in oddsum()

```
function oddsum ( n )
{
    sum = 0
    term = 1
    for (i=1; i<=n; ++i) {
        sum += term
        term += 2
    }
    return sum
}
BEGIN {
    n = 10
    sum = 0
    for (i=1; i<=n; ++i) {
        print "Calling oddsum(" i ")"
        sum += oddsum(i)
        print "sum = " sum
    }
    print sum
}
```

```
$gawk   -f oddsum.awk
Calling oddsum(1)
sum = 2
Calling oddsum(3)
sum = 18
Calling oddsum(5)
sum = 50
Calling oddsum(7)
sum = 98
Calling oddsum(9)
sum = 162
162
```

Fixing the problem

```
function oddsum ( n, i, sum )
{
    print "oddsum(" n ") called"
    sum = 0
    term = 1
    ….. (same as before)
    return sum
}

BEGIN {
    n = 10
    sum = 0
   for (i=1; i<=n; ++i) { sum += oddsum(i) }
    print sum
}
```

```
$gawk  -f oddsum-fixed.awk
oddsum(1) called
oddsum(2) called
oddsum(3) called
oddsum(4) called
oddsum(5) called
oddsum(6) called
oddsum(7) called
oddsum(8) called
oddsum(9) called
oddsum(10) called
385
```

Writing to file: use redirection

```
{
     dept = substr($5, 3, 2)
     if (dept == "MA" ) {
          printf("%s %s %s with roll no. %s  has taken courses %s\n",
$1, $2, $3,  $5, $6) >> "mathdetails.txt"
          nst++;
     }
}
END {
     printf("Number of Math dept. students are %d\n", nst);
}
```

- No need to open the file for writing or close it
- >> is used for appending to the file

```
$gawk -F: -f redirection.awk  student.txt
Number of Math dept. students are 4
$ cat mathdetails.txt
Abhik  Das with roll no. 10MA20012  has taken courses Algorithms,OS,Networks,ML
Arvind  Srinivasan with roll no. 10MA60012  has taken courses Algorithms,OS,ML
Gautam  Kumar with roll no. 10MA60024  has taken courses Algorithms,AI
Anusha V Pillai with roll no. 09MA10001  has taken courses Networks,AI
```

- > means overwrite
- >> means append
- The mode is determined by the first print statement
- After that, there is no distinction between > and >>
- The output filename is to be quoted, otherwise this would be treated as a variable.

Practice Problems

Write gawk programs to:

1. Print the names and roll no.s of all students who have a middle name also
2. Print the name (first and last only) and roll no. of all students who have taken ML
3. Print the name (first and last only) and roll no. of all PG students who have taken ML
4. Print the name and roll no. of all students who have scored more than 75% in OS
5. Print the average marks obtained in Algorithms among all students who have taken the course
6. Print the roll numbers only of all students whose total marks in the courses they have taken is more than 80% (of the total possible marks)
7. Print the name of all students grouped by their student type (UG/PG/RS). All RS student names should be printed first, then all PG students, then all UG students
8. Print the names of all students whose last names has at least 6 characters in it