Scripting Languages

Module 8

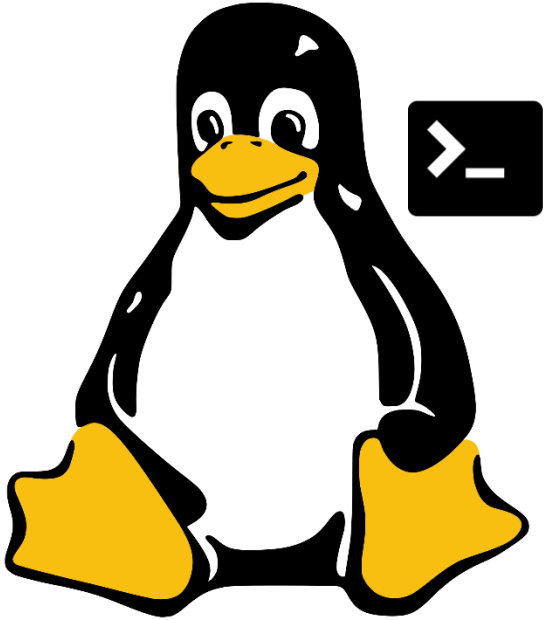
Manipulating Data Using Awk

Contents

Learning Objectives

After finishing this module, you should be able to:

- Execute scripts that use AWK
- Process streams using either sed or AWK or a combination of the two
- Execute scripts that use AWK to process strings and real numbers
- Use AWK in conjunction with other commands such as grep and sed, and utilising regular expressions to produce required solutions



Introduction to **awk**

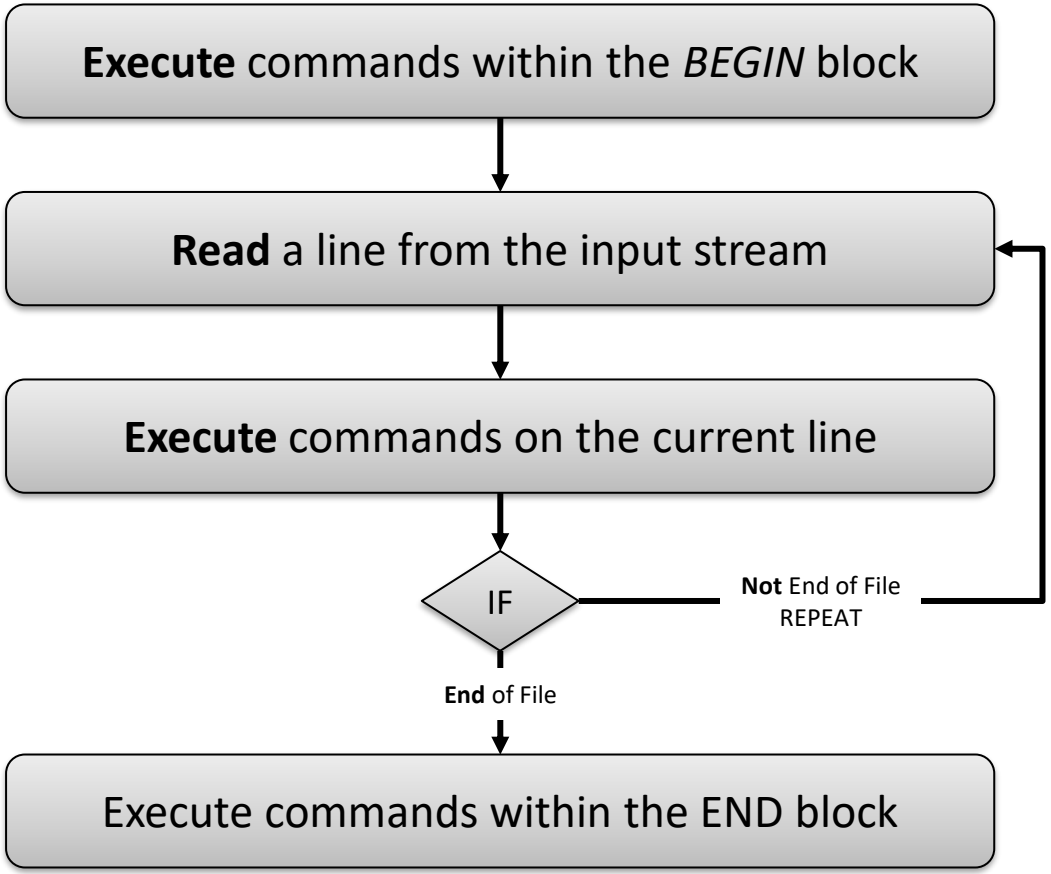
awk

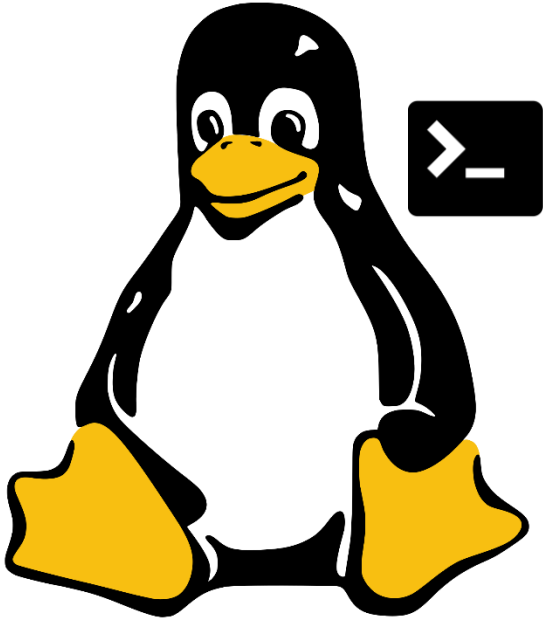
- AWK is a highly-versatile utility that can be used to perform a wide range of useful tasks within the bash environment, including:
  - ✓ Text processing,
  - ✓ Producing formatted text reports,
  - ✓ Performing arithmetic operations,
  - ✓ Performing string operations
- Unlike other key utilities such as sed and grep, AWK is actually an entire programming language in its own right, in which you can:
  - ✓ Define variables
  - ✓ Use string and arithmetic operators
  - ✓ Use control flow and loops

awk History

- AWK was first developed by bell labs in the 1970s
- It was named after the three programmers who originally designed it Alfred Aho, Peter Weinberger and Brian Kernighan (AWK)
- Over the years there have been many different implementations of awk, but the two most common forms in use today are:
  - gawk (used in linux based systems)
  - BWK (used in bsd and MacOS based systems)

The AWK workflow

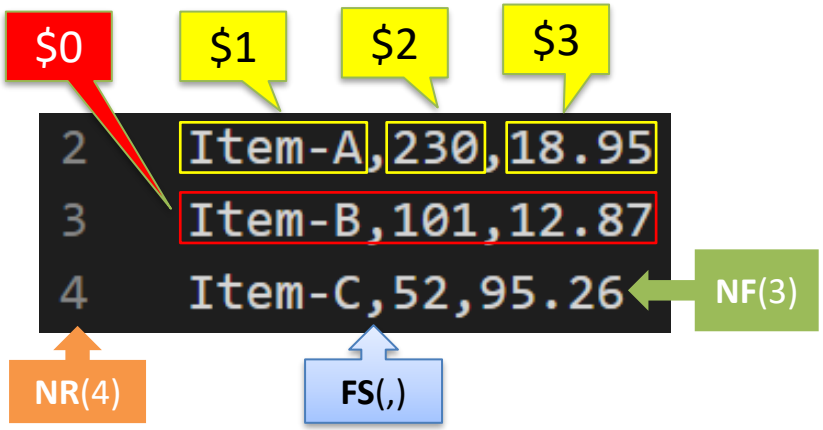


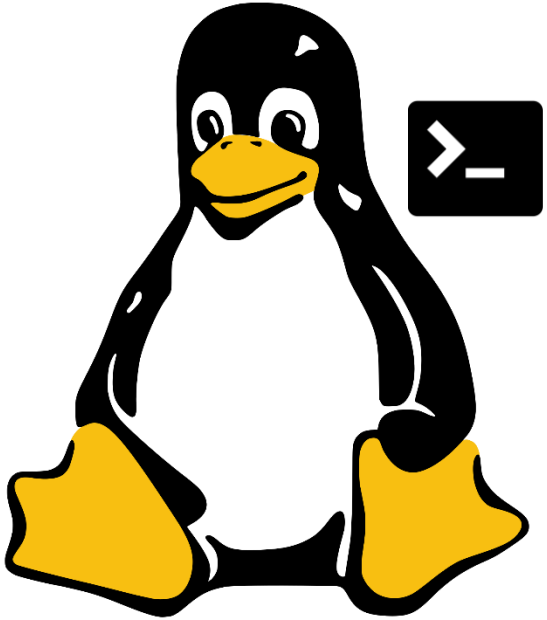


awk default variables

AWK Default Variables

Variable	Purpose
$0	Entire record
$1	Field #1 in record
$2	Field #2 in record
$3	Field #3 in record
...and so on	
NF	Stores # fields in record
NR	Stores line number of each record
FS	File separator





awk Structure

The AWK Structure

```
awk BEGIN {awk-commands}
/pattern/ {commands}
END {awk-commands}
```

■ *optional elements*

```
Item,Units,Price,Tax Status
Hard Disk Drive,230,18.95,T
SSD,101,12.87,T
Printer,52,95.26,T
Mouse Mat,400,5.95,E
DELL Laptop,20,665.30,T
Tablet Cover 14#,154,15.40,E
Printer Cartridge,302,32.40,E
USB 64 GB,220,23.20,T
```

We will use the data in **salesdata.csv** to demonstrate **awk** in action

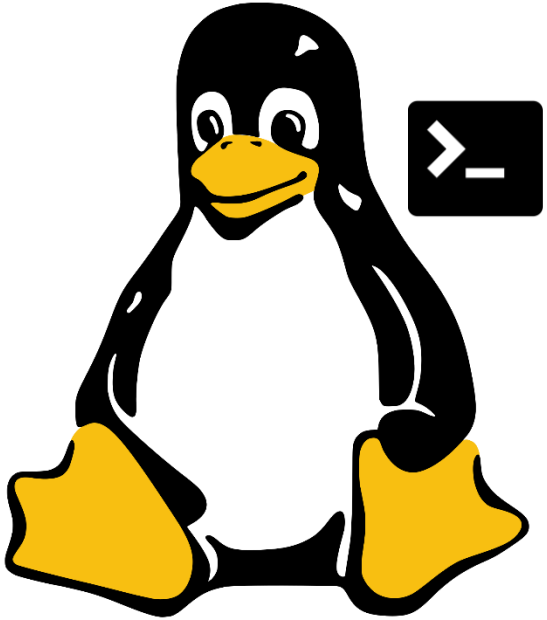
The awk pattern-command structure

```
#!/bin/bash

awk 'BEGIN {print "Q1 SALES REPORT:"}
     NR>1 { print "  " NR "  " $0 }
     END {print "END OF REPORT"}' salesdata.csv

exit 0
```

1 2 3 4

```
vbrown@LAPTOP-N6EFE714:~/scriptlang/w
Q1 SALES REPORT:
  2  Hard Disk Drive,230,18.95,T
  3  SSD,101,12.87,T
  4  Printer,52,95.26,T
  5  Mouse Mat,400,5.95,E
  6  DELL Laptop,20,665.30,T
  7  Tablet Cover 14#,154,15.40,E
  8  Printer Cartridge,302,32.40,E
  9  USB 64 GB,220,23.20,T
END OF REPORT
```

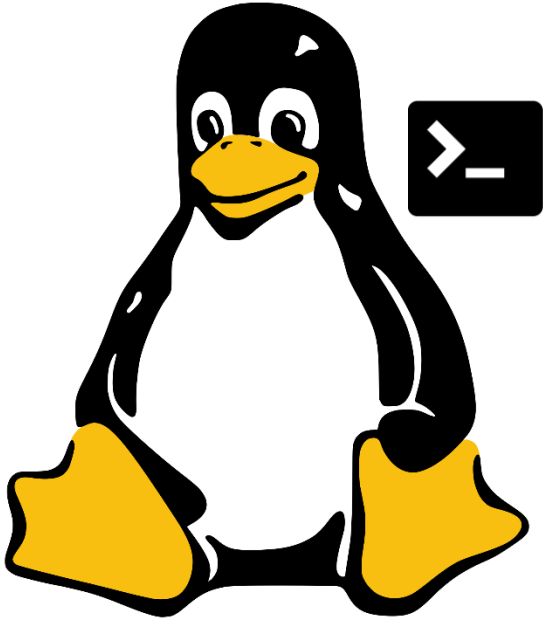
1. BEGIN pattern: Actions awk will execute **once** *before* any input lines are read. The BEGIN pattern is optional.
2. MAIN PATTERN/ACTION: Specifies what is to be done with each line, or specific fields within each line contained with the data handed to awk. One or more statements within curly braces { } are compulsory.
3. END pattern: Actions awk will execute **once** *after* any input lines are read. The END pattern is optional.
4. The DATA SOURCE: Specified **after** the *close* }' of the awk structure <u>if</u> data has not be piped through prior from another process



awk Formatting

printf format codes

Code	Result
%s	String
%5s	String with a minimum of 5 characters
%f	Number (floating point)
%5f	Number with a minimum of 5 characters
%5.2f	Number with a minimum of 5 characters and 2 decimal places
%d	Whole number (Decimal Integer)
%c	Single Character



awk in action

Setting the field separator and output format

```
1   #!/bin/bash
2
3   # Line 7: Use optional BEGIN pattern to set field seperator (FS) to comma and print a header
4   # Line 8: Use printf to set output column widths, then specify fields to be output to terminal
5   # Line 9: Use optional END pattern to print a footer; finish by declaring data source
6
7   awk 'BEGIN {FS=","; print "Q1 SALES REPORT:"}
8       { printf "  "; printf "%-20s %-10s %-10s \n",  $1, $2, $3 }
9       END {print "END OF REPORT"}' salesdata.csv
10
11  exit 0
```

1 2

```
vbrown@LAPTOP-N6EFE714:~/scriptlang/workshops
Q1 SALES REPORT:
  Item                 Units      Price
  Hard Disk Drive      230        18.95
  SSD                  101        12.87
  Printer              52         95.26
  Mouse Mat            400        5.95
  DELL Laptop          20         665.30
  Tablet Cover 14#     154        15.40
  Printer Cartridge    302        32.40
  USB 64 GB            220        23.20
END OF REPORT
```

1. The FS=“,” pattern tells awk that the field separator in the data source is a comma
2. The `%-20s %-10s %-10s \n` pattern provides awk with the formatting pattern of the field output commands that immediately follow

Specifying records and inserting string info

```bash
#!/bin/bash

# Line 7: NR>1 allows record (line) 1 in the data source to be skipped, e.g. because it's a header
# Also note that string info is encapsulated within quotes and field identifiers sit outside of them

awk 'BEGIN {FS=","; print "Q1 PRICE LIST:"}
    NR>1{ print "  "$1"s sells for $"$2" each." }
     END {print "END OF PRICE LIST"}' salesdata.csv

exit 0
```

1

2

```
vbrown@LAPTOP-N6EFE714:~/scriptlang/worksh
Q1 PRICE LIST:
  Hard Disk Drives sells for $230 each.
  SSDs sells for $101 each.
  Printers sells for $52 each.
  Mouse Mats sells for $400 each.
  DELL Laptops sells for $20 each.
  Tablet Cover 14#s sells for $154 each.
  Printer Cartridges sells for $302 each.
  USB 64 GBs sells for $220 each.
END OF PRICE LIST
```

1. The NR>1 test tells awk to *skip* the first record (line) in the source file, in this case because it is an unwanted header

2. Also note that string info is *encapsulated* within quotes “ “ and field identifiers, e.g. $1 $2 sit outside of them

Declared variables and output formatting

```
8   awk 'BEGIN {FS=","; print "PRODUCTS SOLD:"}
9       NR>1{ stotal=$2*$3;
10      stax=stotal*0.1;
11      printf "Total Sales for " $1 " is $%.2f", stotal;
12      printf " (inc. GST of $%.2f", stax;
13      printf ")\n" }
14      END {print "END OF FILE"}' salesdata.csv
15
16  exit 0
```

1

2

```
vbrown@LAPTOP-N6EFE714:~/scriptlang/workshops/ws $ ./awk1.sh
PRODUCTS SOLD:
Total Sales for Hard Disk Drive is $4358.50 (inc. GST of $435.85)
Total Sales for SSD is $1299.87 (inc. GST of $129.99)
Total Sales for Printer is $4953.52 (inc. GST of $495.35)
Total Sales for Mouse Mat is $2380.00 (inc. GST of $238.00)
Total Sales for DELL Laptop is $13306.00 (inc. GST of $1330.60)
Total Sales for Tablet Cover 14# is $2371.60 (inc. GST of $237.16)
Total Sales for Printer Cartridge is $9784.80 (inc. GST of $978.48)
Total Sales for USB 64 GB is $5104.00 (inc. GST of $510.40)
END OF FILE
```

Line 9: Declare a variable named **stotal** to which *product* of field $2 and field $3 is assigned

Line 10: Declare a variable named **stax** to which *10% of stotal* is assigned

Line 11: Print product name and its sales total to terminal formatted to two decimal places

Line 12: Print GST component to terminal formatted to two decimal places

Note:

1. The `%.2f` code formats the value that follows, i.e. `stotal` to two (2) decimal places
2. 'Programmer declared' awk variables do **not** use the *$* prepend, either when declared or when used in awk command sequences

Identifying specific records based on a regex

```
7   awk 'BEGIN {FS=","; print "PRODUCTS SOLD:"}
8       $1 ~ /^P/ { stotal=$2*$3;
9       stax=stotal*0.1;
10      printf "Total Sales for " $1 " is $%.2f", stotal;
11      printf " (inc. GST of $%.2f", stax;
12      printf ")\n" }
13      END {print "END OF FILE"}' salesdata.csv
14
15  exit 0
```

1

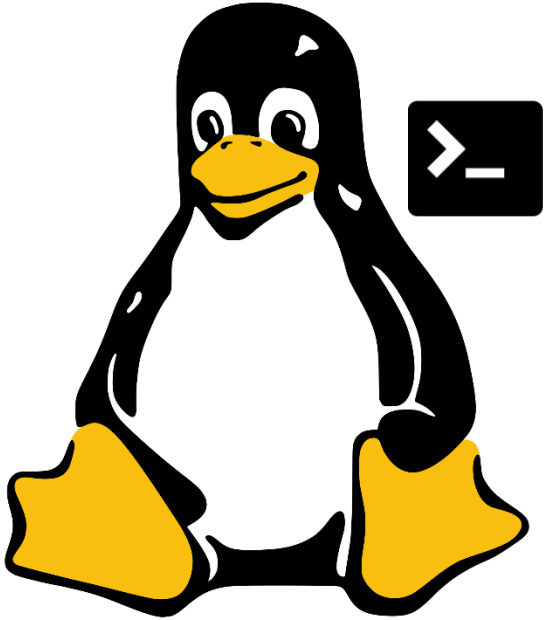
2

```
vbrown@LAPTOP-N6EFE714:~/scriptlang/workshops/ws8$ ./awk1.sh
PRODUCTS SOLD:
Total Sales for Printer is $4953.52 (inc. GST of $495.35)
Total Sales for Printer Cartridge is $9784.80 (inc. GST of $978.48)
END OF FILE
```

Line 8: Declares a pattern that determines which records (lines) will be acted upon by awk commands. The pattern in this case is those records (lines) in which field **$1** *starts* with uppercase *P*. The pattern is placed immediately **before** awk's *main command block*, i.e. just before the opening {

Note:

1. The ~ symbol in the context means contains. !~ inverts this, i.e. does **not** contain
2. In awk, the regular expression pattern is delimited by forward slashes /^P/



awk Conditional Statements

Using an IF control structure in awk

```
3   awk 'BEGIN {FS=","; print "SALES REPORT:"}
4       NR>1 { if ($4=="T")
5               {
6                   stotal=$2*$3;
7                   stax=stotal*0.1;
8                   printf "Total Sales for " $1 " is $%.2f", stotal;
9                   printf " (inc. GST of $%.2f", stax;
10                  printf ")\n";
11              }
12          else
13              {
14                  stotal=$2*$3;
15                  printf "Total Sales for " $1 " is $%.2f", stotal;
16                  printf " (GST Exempt)\n";
17              }
18      }
19      END {print "END OF FILE"}' salesdata.csv
20
21  exit 0
```

1 2 3 4 5

```
vbrown@LAPTOP-N6EFE714:~/scriptlang/workshops/ws8$ ./awk1.sh
SALES REPORT:
Total Sales for Hard Disk Drive is $4358.50 (inc. GST of $435.85)
Total Sales for SSD is $1299.87 (inc. GST of $129.99)
Total Sales for Printer is $4953.52 (inc. GST of $495.35)
Total Sales for Mouse Mat is $2380.00 (GST Exempt)
Total Sales for DELL Laptop is $13306.00 (inc. GST of $1330.60)
Total Sales for Tablet Cover 14# is $2371.60 (GST Exempt)
Total Sales for Printer Cartridge is $9784.80 (GST Exempt)
Total Sales for USB 64 GB is $5104.00 (inc. GST of $510.40)
END OF FILE
```

1. Logical test is encapsulated within started parenthesis
2. Assignations differ from shell script, more like PHP and others
3. Command must be enclosed in { } (unless only one command)
4. Commands must be terminated with semi-colon ;
5. If structure does not require close statement, e.g. endif

Note:
If more than two (2) options apply, then `else if (condition-x)` is used

A more complex awk example

```
3   awk 'BEGIN {FS=",";
4               print "SALES REPORT:";
5               gross=0;
6               gstcomp=0;
7               net=0;
8               }
9     NR>1 {
10        if ($4=="T")
11            {
12                stotal=$2*$3;
13                stax=stotal*0.1;
14                printf "Total Sales for " $1 " is $%.2f", stotal;
15                printf " (inc. GST of $%.2f", stax;
16                printf ")\n";
17                gross=gross+stotal;
18                gstcomp=gstcomp+stax
19            }
20         else
21            {
22                stotal=$2*$3;
23                printf "Total Sales for " $1 " is $%.2f", stotal;
24                printf " (GST Exempt)\n";
25                gross=gross+stotal;
26            }
27     }
28     END {
29         net=gross-gstcomp;
30         printf "Gross Sales: $%.2f", gross;
31         printf "\n";
32         printf "GST Component: $%.2f", gstcomp;
33         printf "\n";
34         printf "Net Sales: $%.2f", net;
35         printf "\n" }' salesdata.csv
```

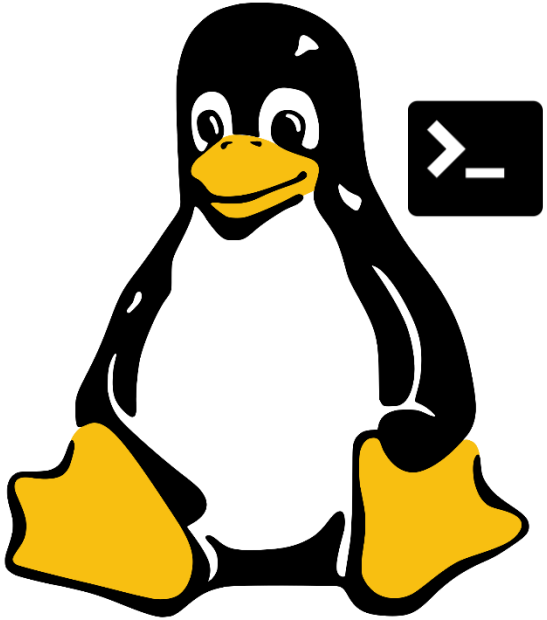
1

2

```
vbrown@LAPTOP-N6EFE714:~/scriptlang/workshops/ws8$ ./awk1.sh
SALES REPORT:
Total Sales for Hard Disk Drive is $4358.50 (inc. GST of $435.85)
Total Sales for SSD is $1299.87 (inc. GST of $129.99)
Total Sales for Printer is $4953.52 (inc. GST of $495.35)
Total Sales for Mouse Mat is $2380.00 (GST Exempt)
Total Sales for DELL Laptop is $13306.00 (inc. GST of $1330.60)
Total Sales for Tablet Cover 14# is $2371.60 (GST Exempt)
Total Sales for Printer Cartridge is $9784.80 (GST Exempt)
Total Sales for USB 64 GB is $5104.00 (inc. GST of $510.40)
Gross Sales: $43558.29
GST Component: $2902.19
Net Sales: $40656.10
```

1. The optional BEGIN statement can be used to *declare* and *initialise* variables that are to be used **across** *all records* collectively, rather than record by record, e.g. `gross`, `gstcomp` and `net`

2. The optional END statement can be used to *calculate totals* **after** all records have been processed for *summary* purposes



awk Functions

Functions

- Functions in awk behave in a similar way to functions in bash
- Just like shell script, awk functions are useful for breaking scripts up into logical modules and reducing the need for repeated code

Function	Purpose
sin()	Sine
cos()	Cosine
tan()	Tangent
sqrt()	Square root
exp()	Exponential
log()	Logarithm
rand()	Random Number Generator
length()	String length
split()	String splitter
toupper()	Convert string to uppercase
tolower()	Convert string to lowercase

Custom awk functions

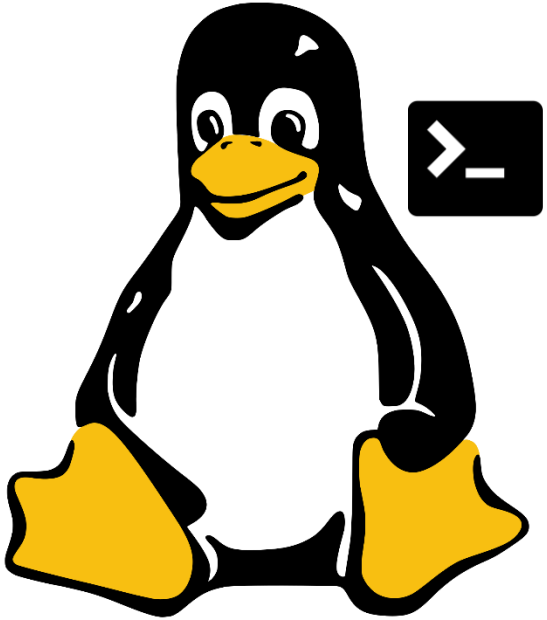
```
3   awk '
4
5   function PrintInYellow(string){
6               printf "\033[0;33m%s\033[0m", string;
7           }
8
9   BEGIN {FS=",";
10              print PrintInYellow("SALES REPORT:");
11              gross=0;
12              gstcomp=0;
13              net=0;
14              }
15      NR>1 {
```

1

2

```
vbrown@LAPTOP-N6EFE714:~/scriptlang/workshops/ws8$ ./awk1.sh
SALES REPORT:
Total Sales for Hard Disk Drive is $4358.50 (inc. GST of $435.85)
Total Sales for SSD is $1299.87 (inc. GST of $129.99)
Total Sales for Printer is $4953.52 (inc. GST of $495.35)
Total Sales for Mouse Mat is $2380.00 (GST Exempt)
Total Sales for DELL Laptop is $13306.00 (inc. GST of $1330.60)
Total Sales for Tablet Cover 14# is $2371.60 (GST Exempt)
Total Sales for Printer Cartridge is $9784.80 (GST Exempt)
Total Sales for USB 64 GB is $5104.00 (inc. GST of $510.40)
Gross Sales: $43558.29
GST Component: $2902.19
Net Sales: $40656.10
```

1. Declare the custom function immediately after awks opening single parenthesis ‘
2. Apply the functions in the awk command body as required



Integrating **awk** in shell scripts

Using awk – Example 1

```
#!/bin/bash

read -p 'Enter float 1: ' fl1
read -p 'Enter float 2: ' fl2

result=$( echo $fl1 $fl2 | awk '{prod=$1*$2; printf"%.2f\n", prod }' )

echo "$fl1 multiplied by $fl2 is $result"

exit 0
```

1 2 3

```
vbrown@LAPTOP-N6EFE714:~/scriptlang/workshops/ws8$ ./awk2.sh
Enter float 1: 3.3
Enter float 2: 6.2
3.3 multiplied by 6.2 is 20.46
```

1. Uses *command substitution* to immediately assign the results of command sequence to a variable named `$result`
2. `Echo` used to provide stored inputs (`$fl1`, `$fl2`) to `awk` via *piping*
3. Then `awk` makes the required calculations, the product of which is stored in `$result`

Using awk - Example 2

```
#!/bin/bash

pre="<tr><td>"
post="<\/td><\/tr>"
mid="<\/td><td>"

cat scores.html | grep "<td>" | sed -e "s/^$pre//g; s/$post$//g; s/$mid/ /g" | awk '{ avg=($2+$3+$4)/3; printf $1 " Average - %.2f\n", avg }'

exit 0
```

1 2 3 4

```
<body>
<h1>Attempts</h1>
</body>
</html>
<table style="width:500px;text-align:left;">
<tr><th style="width:200px;">Attempts</th><th style="width:100px;">
<tr><td>Week1</td><td>27.2</td><td>28.1</td><td>25.6</td></tr>
<tr><td>Week2</td><td>22.0</td><td>27.2</td><td>31.2</td></tr>
<tr><td>Week3</td><td>31.2</td><td>25.55</td><td>24.1</td></tr>
<tr><td>Week4</td><td>19.8</td><td>22.2</td><td>33.3</td></tr>
```

```
vbrown@LAPTOP-N6EFE714:~/scri
Week1 Average - 26.97
Week2 Average - 26.80
Week3 Average - 26.95
Week4 Average - 25.10
```

1. Grab the content of `scores.html` and *pipe* it through to `grep`
2. Use `grep` to eliminate all lines passed to it except those that contain `<td>` and then *pipe* through to `sed`
3. Use `sed` to eliminate all remaining HTML tags, ensuring that the `</td></td>` sequence is replace by a space, then *pipe* the results to `awk`
4. Use `awk` to calculate the averages of the float values on each of the four records that remain and print to the terminal formatted to two (2) decimal places

Using awk – Example 3 (Part A)

```
#!/bin/bash

# This script scans the files in a nominated directory and outputs their size on disk in Mb, Kb or b.

RED='\033[0;31m' # to colour error messages
GREEN='\033[0;32m' # to highlight key output values
BLUE='\033[0;34m' # for output headers
NC='\033[0m' # switches off the application of a colour to oputput

# The getsize() function coverts file sizes in bytes to Kb or Mb where applicable
getsize() {
    let mb=1048576 # number of bytes in a megabyte
    let kb=1024 # number of bytes in a kilobyte
    if [[ $1 -ge $mb ]]; then
        echo "$(echo $1 | awk '{printf "%.2f", $1/1024/1024}')Mb"
    elif [[ $1 -ge $kb ]]; then
        echo "$(echo $1 | awk '{printf "%.2f", $1/1024}')Kb"
    else
        echo "$1b"
    fi
}

# If temp.txt file exists from last run, delete it
if [[ -f temp.txt ]]; then
    rm temp.txt
fi
```

1

2

Item	Explanation
1	A function that will be used to calculate the disk size of a file, formatted appropriately in Mb, Kb or bytes, as applicable
2	Delete the temporary file the script creates in case it is still present from the last run

Using awk – Example 3 (Part B)

```
28  # get the directory and path option from the user
29  echo -e "Please provide path to directory to scan and a path option, e.g. ${GREEN}~/docs f${NC}"
30  echo -e "PATH OPTIONS: ${BLUE}f${NC} -> full path, ${BLUE}c${NC} -> current directory, ${BLUE}s${NC} -> child directory)"
31  read -p "Enter a valid directory and path option: " dir opt
32
33  # if either of the requested values is missing, inform user and exit script with error code
34  if [[ -z $dir ]] || [[ -z $opt ]]; then
35      echo -e "${RED}One or more arguments h ot been provided. Exiting...${NC}" && exit 1
36  fi
37
38  # If dir var contains a value, strip leading and trailing path delimiters
39  dir=`echo $dir | sed 's/^[~/]//' | sed 's/\/$]//'`
40
41  # check that the path/dir provided exists, and if so gn to path variable for later use
42  case $opt in
43      f|F) # if full path option is selected
44          if ! [[ -d ${HOME}${dir} ]]; then # if the path/dir does not exist, inform user and exit scri
45              echo -e "${RED}Directory does not exist. Exiting...${NC}" && exit 1
46          else
47              path=$(echo -n "${HOME}${dir}/*") # else assign valid path to variable with wildcard oper
48          fi
49      ;;
50      c|C|s|S) # if cwd or subdirectory path option is selected
51          if ! [[ -d ${dir} ]]; then # if the path/dir do not exist, inform user and exit script with e
52              echo -e "${RED}Directory does not exist. Exiting...${NC}" && exit 1
53          else
54              path=$(echo -n "${dir}/*") # else assign valid path to variable with wildcard operator
55          fi
56      ;;
57      *) echo -e "${RED}Invalid path option. Exiting...${NC}" && exit 1 # if an invlalid path option is given
58      # inform user and exit script with error code
59  esac
60
```

Item	Explanation
3	If either of the required command line arguments are missing, terminate script and throw error
4	Strip leading and trailing path delimiters if they are present
5	Use a case statement to process the option argument
6	In either case, if the directory path is invalid, or an invalid path option is provided, terminate the script and throw an error

Using awk – Example 3 (Part C)

```
61  echo -e "${BLUE}FILENAME\tFILESIZE${NC}" # print out a header for the results
62  for item in $path # loop through each item in the target directory
63  do
64      if [[ -f $item ]]; then # if a file
65          fname=$(basename $item) # extract basename and assign to var
66          fsize=$(getsize $(du -b $item | cut -f 1)) # get adjusted size of file using custom function
67          echo $fname $fsize | awk '{printf "%-15s %-10s \n", $1,$2}' >> temp.txt # print out the results using awk to a file
68      fi
69  done
70
71  cat temp.txt | sort --ignore-case -k 1 # output the results in temp.txt file ordered a-z (case-insensitive)
72
73  exit 0
```

7 8 9 10

Item	Explanation
7	Use the basename command to strip the path element of the file name
8	Use du command in conjunction with custom function to get file size
9	Use awk to format the results to the a temp file
10	Use sort to output the results to terminal in alphabetical order

Using awk – Example 3 (Results)

```
[redacted] ~/scrlang/lec$ ./dirscan.sh
Please provide path to directory to scan and a path option, e.g. ~/docs f
PATH OPTIONS: f -> full path, c -> current directory, s -> child directory)
Enter a valid directory and path option: ~/docs f
FILENAME        FILESIZE
accounts.pdf    134.46Kb
bash.png        28.09Kb
inventory.pdf   245.35Kb
LawNotes.docx   96.91Kb
linux.jpg       186.87Kb
logfile.csv     38.77Kb
mint.png        61.00Kb
vals.sh         132b
ws_guide.pdf    1.68Mb
```

```
[redacted]:~/scrlang/lec$ ./dirscan.sh
Please provide path to directory to scan and a path option, e.g. ~/docs f
PATH OPTIONS: f -> full path, c -> current directory, s -> child directory)
Enter a valid directory and path option: ~/docs
One or more arguments have not been provided. Exiting...
[redacted]:~/scrlang/lec$ ./dirscan.sh
Please provide path to directory to scan and a path option, e.g. ~/docs f
PATH OPTIONS: f -> full path, c -> current directory, s -> child directory)
Enter a valid directory and path option: ~/docs p
Invalid path option. Exiting...
[redacted]:~/scrlang/lec$ ./dirscan.sh
Please provide path to directory to scan and a path option, e.g. ~/docs f
PATH OPTIONS: f -> full path, c -> current directory, s -> child directory)
Enter a valid directory and path option: ~/yada f
Directory does not exist. Exiting...
```

Using awk – Example 4

```
#!/bin/bash

# This script scans through a web access log file for matches to user inputs

if [[ -f temp.csv ]]; then
    rm temp.csv
fi

if [[ -f results.csv ]]; then
    rm results.csv
fi

while true; do
    read -p "Please enter name of log file (.csv) and protocol (TCP,UDP,ICMP or GRE): " file prot # get the required values from user
    if ! [[ -f $file ]]; then # check the stipulated file actually exists
        echo "No such file in this directory. Please try again." # if not, user must try again
    elif ! [[ `echo $prot | tr [a-z] [A-Z]` =~ ("TCP"|"UDP"|"ICMP"|"GRE") ]]; then # Not a valid protocol...
        echo "Invalid protocol. Select from TCP,UDP,ICMP or GRE. Please try again." # So user must try again
    else
        cat $file | sed 's/ *,/,/g' > temp.csv # remove any trailing spaces that may exist in the .csv file
        break # else, file and protocol are valid so move on
    fi
done

if [[ $(cat temp.csv | grep -ic "$prot") -gt 0 ]]; then # if the search gets results...
    cat temp.csv | grep -i "$prot" | awk 'BEGIN {FS=","} { printf "%-10s %-10s %-10s \n", $3,$8,$9 }' | sort -r -n -k 2 > results.csv
    echo "$(cat results.csv | wc -l) matches were found of which $(cat results.csv | uniq | wc -l) are unique, these being:"
    cat results.csv | uniq
else
    echo "No matches found" # else just tell the user no matches found
fi

exit 0
```

1

2

3

4

5

6

Using awk – Example 4 (Explained and Results)

Item	Explanation
1	If the temporary files the script creates are still present from the last run, delete them
2	Using an infinite loop, validate the file name and protocol entered by the user, and if either invalid, inform them of error and prompt them to try again
3	If inputs are valid, remove any trailing spaces in any of the web log's fields, which can often be present in such files, and transfer the fixed results to a temp file
4	If matches are found...
5	If the temp file contains matches for the protocol entered by the user, write them to a results file, formatted with awk and sorted by one of the numeric fields, e.g. packets
6	Display only unique results to terminal

```
[redacted] ~/scrlang/lec$ ./searchsort.sh
Please enter name of log file (.csv) and protocol (TCP,UDP,ICMP or GRE): logfile.csv udp
7 matches were found of which 4 are unique, these being:
UDP        1          76
UDP        1          46
UDP        1          437
UDP        1          122
[redacted] ~/scrlang/lec$ ./searchsort.sh
Please enter name of log file (.csv) and protocol (TCP,UDP,ICMP or GRE): logfile.csv
Invalid protocol. Select from TCP,UDP,ICMP or GRE. Please try again.
Please enter name of log file (.csv) and protocol (TCP,UDP,ICMP or GRE): logfile.csv yada
Invalid protocol. Select from TCP,UDP,ICMP or GRE. Please try again.
Please enter name of log file (.csv) and protocol (TCP,UDP,ICMP or GRE): myfile.csv tcp
No such file in this directory. Please try again.
Please enter name of log file (.csv) and protocol (TCP,UDP,ICMP or GRE): logfile.csv icmp
5 matches were found of which 5 are unique, these being:
ICMP       1          57
ICMP       1          56
ICMP       1          465
ICMP       1          150
ICMP       1          104
```

Terms to Review and Know

- regex in awk
- If statements in awk
- Functions in awk
- awk scripts
- awk formatting
- parsing
- fields
- records
- printf