Jack Berberian

CIS 245

Awk Assignment

2/5/2022

What is Awk?

Similar to sed, awk is a power tool for Linux.

Glossary of Terms

“\t” is used a ‘spacer’ inside of the { } brackets.

-F’[enterdelimiterinsidehere]’ is how a delimiter is signified

The last part of an awk command needs to be the file that will be manipulated. In this case, it’s AwkLab.data but it can be called anything.

How to Use Awk

1. Print all the First Names

The command we need to enter here is: awk ‘{print $1}’ AwkLab.data

What this command does is tell awk to print the first field it finds in the file AwkLab.data which is the name in this case.

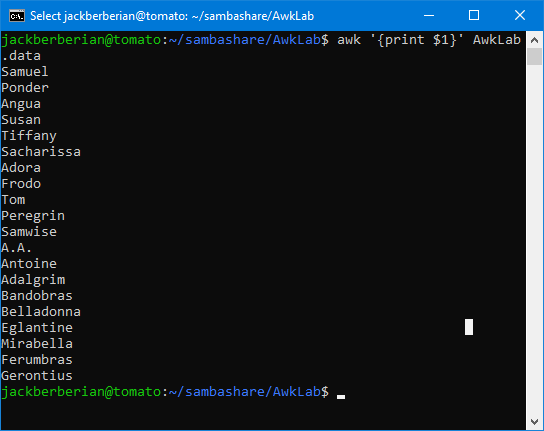
Sample Line: Samuel Vimes:(510) 548-1278:250:100:175

Here, $1 is the identified as the first field because there’s no delimiter or other signifier used to tell Awk that the line starts somewhere else.

But by using : as a delimiter, Awk prints out everything before it as the first field. By changing the delimiter by using -F‘[:]’, then you can change the field values. The delimiter is inside of the brackets in -F’[]’

Text

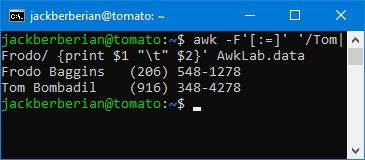
Description automatically generated



1. Print phone numbers for Tom and Frodo after their names

The command that we need to enter here is: awk -F'[:=]' '/Tom|Frodo/ {print $1 "\t" $2}' AwkLab.data

What this does is pattern match for Tom and Frodo and once it finds a match for those lines, it prints their names as $1 and the phone number as $2 since it’s using the semicolon as a field separator.



If you wanted to search for someone else’s phone number, all you’d have to do is change the words Tom and Frodo in the dash marks.

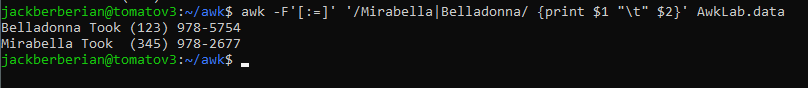
Text

Description automatically generated

awk -F'[:=]' '/Tom|Frodo/ {print $1 "\t" $2}' AwkLab.data

instead of searching for Tom and Frodo,

awk -F'[:=]' '/Mirabella|Belladonna/ {print $1 "\t" $2}' AwkLab.data



Will search for the names and phone numbers of the fields that pattern match Mirabella and Belladonna.

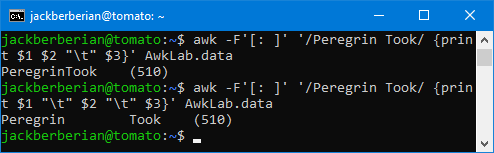
Awk -F’[:=}’ ‘/enterwordhere|enterwordhere/ {print $1 “\t” $2}’ Awklab.data is an awk command that uses := as a delimiter, then pattern matches for what matches enterwordhere, and then prints the $1 and $2 values of the fields in the lines that do match using the file AwkLab.data

1. Print Peregrin’s full name and phone number area code only.

The command that we need to enter here is: awk -F'[: ]' '/Peregrin Took/ {print $1 $2 "\t" $3}' AwkLab.data

What this does is pattern match for Peregrin, similar to the previous lines and then prints their full name and phone number area code only.

Explanation: Here, Awk uses the semicolon and a space as a field separator, and then pattern matches for any lines that contain Peregrin Took. Since there are other Tooks in the file and we specifically want Peregrin, we have to enter ‘/Peregrin Took/ or else we could get undesired results. The $1 is representing her first name, the $2 is representing her last name, and the $3 is representing the area code of her phone number. If Peregrin was changed to Frodo Baggins, it would replace Peregrin’s results with Frodo’s own. Also, the “\t” is a spacer.



Text

Description automatically generated

With this image, you can see how to use this for other applications. By changing the name Peregrin, Awk goes through each line of the file looking for just Peregin, and then if it finds a match, it prints the $1, $2, and $3 values which because of the delimiter, are the first, last name and phone number area code only.

1. Print all phone numbers in the 123 area code along with the names.

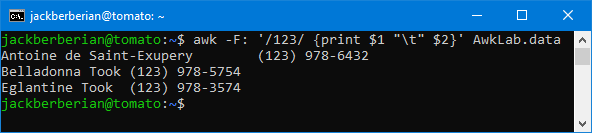
The command that we need to enter here is: awk -F: ‘/123/ {print $1 “\t” $2}’ AwkLab.data

What this does is pattern match for lines with 123, which would be the area code that we’re looking for and then it prints the first and second fields it finds which are the name and phone number of the people.

Explanation: What this does is also use the semicolon as a field separator and pattern match for lines that contain the numbers 123 which would be the area code. A smarter way to search for 123 area codes would be to use ‘/(123)/ in place of just ‘/123/ because it could also return 3 digit numbers elsewhere in the file that aren’t the area code which wouldn’t be a problem if the parenthesis are used indicating its an area code. In other words, ‘/978/ could return 978 that occurs anywhere in the file instead of the area code.

Text

Description automatically generated



1. **Print all Last Names beginning with either a T or D**

Command to enter: awk -F'[:]' '{print $1}' AwkLab.data | awk '$NF ~ /^[TD]/ {print $NF}'

A screenshot of a computer

Description automatically generated

Explanation:

Here in the first part of the command, Awk searches and prints the first and last names of everybody in the file while we only want last names that begin with either a T or D. The semicolon is again used as a field separator and the $1 represents the first and last name. The | is a symbol called a pipe. In a sense, a pipe (|) could be described as what the computer sees. Down below in the screenshot, this is the data the computer sees from the running of the command and what we humans see. Therefore if we were to pipe this, it’d be what was on the screen.

A screenshot of a computer

Description automatically generated

Lastly, what this does is pipe the output from the first half of the command and match for fields that start with T or D. We only want last names that start with T or D so $NF which signifies the field matches (~) lines that only start with T and D and then prints it.

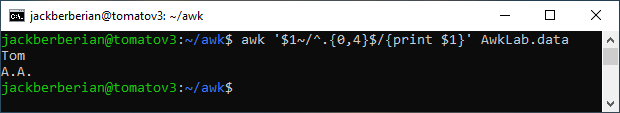
A screenshot of a computer

Description automatically generated

1. Print all first names containing four or less characters

The command that we need to enter here is: awk ‘$1~/^.[0,4]$/{print $1}’ AwkLab.data

What this does is search in the first field on each line for fields that have less than or equal to 4 characters and then quits because of the $ which signifies the end of the line and then prints what it found.



Explanation: The first name is the field $1 in this case. Inside the ‘’, if the $1 field only contains from 0 to 4 characters, it’ll print $1 (the first name), otherwise it will not. You can modify it so it’ll look for first names from 0-9, 0-5 by just changing the 0 and the 4.

1. Print the first names and area codes of all those in the 916 area code.

The command we need to enter here is: awk -F’[: ]’ ‘/(916)/ {print $1 “\t” $2}’ AwkLab.data

What this does is search for lines that match 916 which is the area code we’re looking for and then prints $1, $2 which is the first name and phone number.

A screenshot of a computer

Description automatically generated

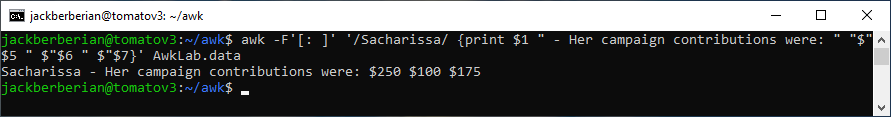
Explanation:

In short, Awk is only printing the $1 and $2 fields for lines that contain (916) for an area code. Since 916 could appear elsewhere in the file, we add parenthesis as is the standard format for an area code to make sure we don’t get any undesired results. In this case, since the field separator is a semicolon and a space, the $1 field this time is just the name, and the $2 field is the last name, while the $3 is just the area code. Since we don’t want the last name, we just tell awk to print $1 and $3 which is just the first name and area code. If we changed (916) to (978), it’d print out people who have an area code of (978) instead.

1. Print Sacharissa’s campaign contributions following her name. Each value should be printed with a leading dollar sign; e.g., $250 $100 $175

The command that we need to enter here is: awk -F'[: ]' '/Sacharissa/ {print $1 " - Her campaign contributions were: " "$"$5 " $"$6 " $"$7}' AwkLab.data

What this does is use a semicolon and a space as a field separator for lines that have Sacharissa. Once it finds a line that does have it, it prints $1 which is her name, and then $%, $6, and $7 which are her campaign contributions.



Explanation: What this does is similar to the other questions in the file. It only searches for lines that have Sacharissa and when it does it prints the $5, $6, and $7 fields which are the campaign contributions for her every 3 months. If we changed Sacharissa to Frodo to Tiffany instead, it would print out her campaign contributions instead.

1. **Print last names followed by a comma and the phone number. Be careful of the last name’s format.**

awk -F'[:]' '{print $1","$2}' AwkLab.data | awk -F' ' '{print $(NF-1),$NF}'A screenshot of a computer

Description automatically generated

Explanation: What this does is print last names followed by a comma and then the phone number. Similar to a previous problem above, this also pipes the output back to Awk again. At first, Awk only prints the whole name and the phone number, we only want the last name. What the pipe command does is subtract the first name field using the NR command which is the number fields and then prints the output which is just the last name and phone number.

Text

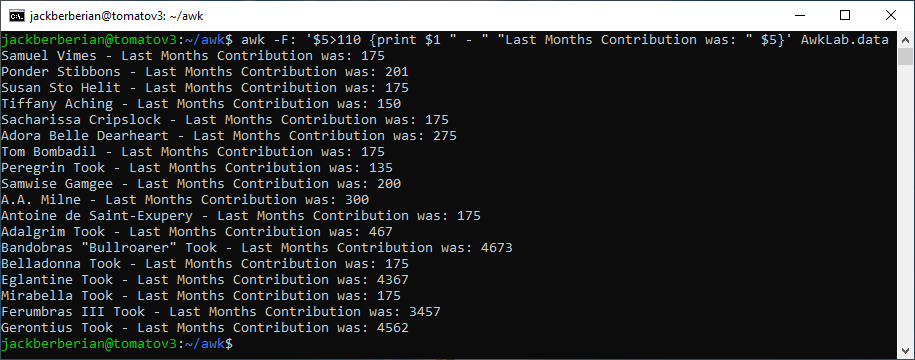
Description automatically generated

Text

Description automatically generated

1. Print the first and last names of those who contributed more than $110 in the last month. Make sure to include their last month contribution amount after the name.

The command we need to enter here is: awk -F: '$5>110 {print $1 " - " "Last Months Contribution was: " $5}' AwkLab.data

What this does is search lines that have a $5 field that’s greater than 110 and then prints it following their name.

Explanation: What this does is search for every that has a last month contribution greater than $110. With the field separator being a semicolon again, the last months contribution is the $5 in this case. By settings $5>110, it’ll only print the $5 field if it’s greater than 110. If we want to look for contributions of greater than 200, 500, it’s as easy as just changing the 110. Or the $5 is also changeable.

1. Print the last names, phone numbers, and first month contribution of those who contributed less than $150 in the first month.

The command that we need to enter here is: awk -F: '$3<150{print $1 "\t" "Their contribution was: " $3}' AwkLab.data

What this means is that it searches for lines that have a $3 field that’s less than 150 and the prints them out.

A screenshot of a computer

Description automatically generated

Explanation:

What this does which is similar to above, is print the $3 field which is the first month’s contribution in this case only if it’s less than 150. The $1 field is the person’s name and the $2 is their phone number while the $3 field is their contribution. If you change the $3<150 part as well as the $3 in the {}, you can obtain different outputs, such as second month’s contribution over $500 as well.

1. **Print the first names and contribution of those who contributed between $75 and $150 in the first month.**

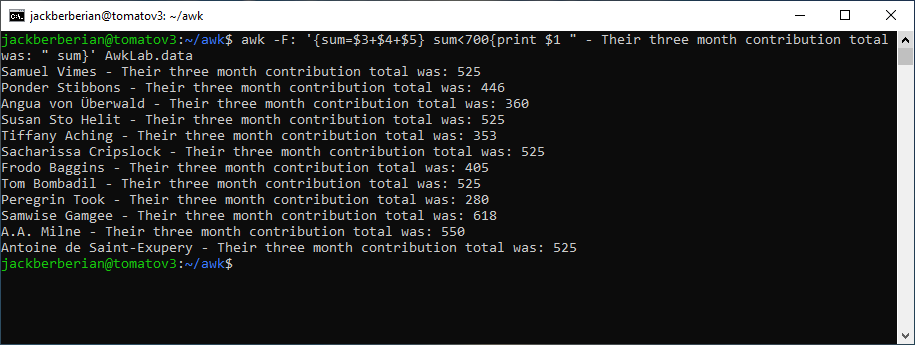
**Command: awk -F: '{if ($3 <= 150 && $3 > 75) {print $1 " - " $3}}' AwkLab.data**

**Explanation: What this is supposed to do is just find the people whose first month’s contribution or their $3 field value is equal to greater than 75 and less than 150 in the first month. Similar to the previous questions, this one looks for a field value $3 that’s less than or equal to 150 and greater than 75. Awk -F: ‘{print $1 “ - “ $3}’ prints out the value of $3 for everybody. As you can see these are all the $3 values. The first command does not print out anything because there is no value from 75-150. The && is a Boolean operator for AND so if $3 is less than or equal to 150 and greater than 75, it’ll print out $3 but since there is no value, it doesn’t print anything. By changing the value of 150 to 175, it prints out two lines successfully.**

**A screenshot of a computer

Description automatically generated**

1. Print the first and last names and total contributions of those who contributed less than $700 over the three-month period.

awk -F: '{sum=$3+$4+$5} sum<700{print $1 " - Their three month contribution total was: " sum}' AwkLab.data

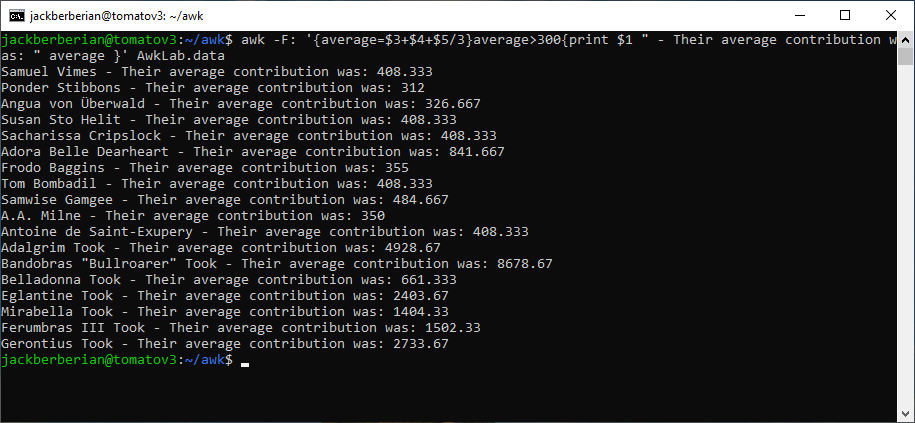
Explanation: What this does is print the first, last name, and total contribution of those who donated less than $700. In this case, $3,$4,$5 are the field values for each month and the sum= equation, calculates the total sum for each person and sets it to equal sum. The by entering sum<700, Awk will only print the sum value if it’s less than 700. If it’s greater than 700, it’ll ignore it.

1. Print the first names and first letter of the last name, and average contribution of those who had an average contribution of more then $300

The command that we need to enter here is: awk -F: '{average=$3+$4+$5/3}average>300{print $1 " - Their average contribution was: " average }' AwkLab.data

What this does is create a variable called average with the values of all three months contributions and then average them by 3 for the average.

Explanation: what this does is similar to the one above. We only want to find the people who had an average contribution of more than 300 so we have to add up the months and average them. By entering average=$3+$4+$5/3}average>300, Awk adds up and sets the three months contribution for each person to average and then divides it by 3. Then if average is greater than 300, Awk will print it.



1. Print the last name of those not in the 916 area code.

awk -F: '$2 !~/(916)/{print $1 "\t" $2}' AwkLab.data

Explanation: What this does is pattern match for lines that have a $2 field which is the phone number in this case that is not 916 and then prints it. The ! is another way of saying the inverse for Awk. If we changed the (916) to (978), it would print the last name of people not in (978) area code instead.

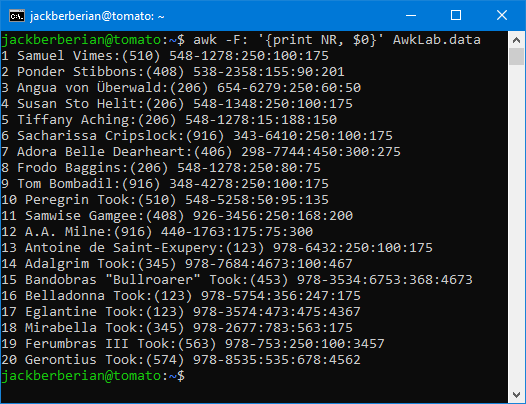
A screenshot of a computer

Description automatically generated with medium confidence

1. Print each record preceded by the number of the record.

The command that we need to enter here is: awk -F: ‘{print NR, $0}’ AwkLab.data

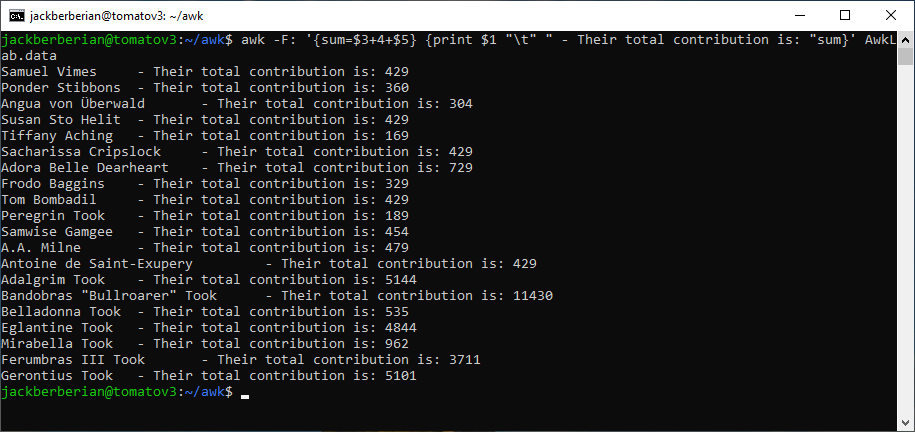
Explanation: What this does is print the line record for the whole line because of the inclusion of NR. The whole line represents the field $0 and NR is the number of records variable which appends the line number to the beginning of the line.



1. Print the name and total contribution of each person.

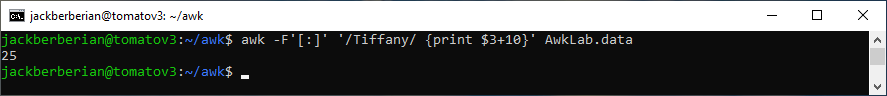
The command that we need to enter here is: awk -F: '{sum=$3+4+$5} {print $1 "\t" " - Their total contribution is: "sum}' AwkLab.data

Explanation: What this means is that awk stores the three month’s contributions in a total called sum and then prints it when sum is entered in the brackets similar to how average worked above as well.



1. **Add $10 to Tiffany Aching's first contribution.**

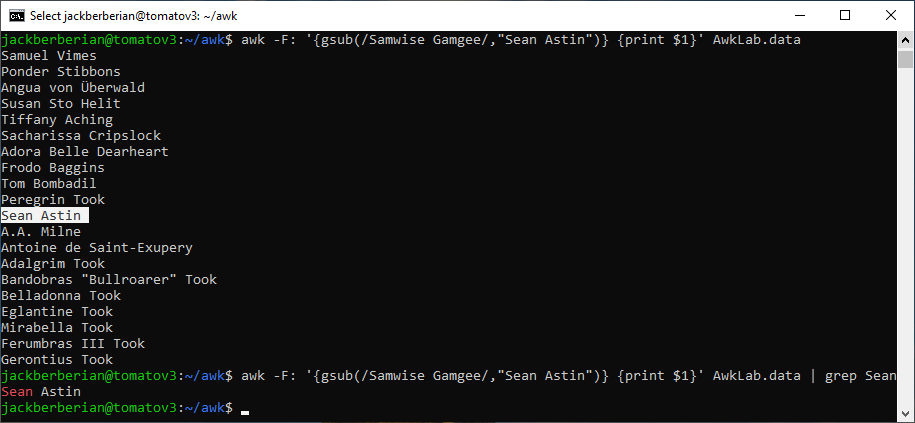
awk -F'[:]' '/Tiffany/ {print $3+10}' AwkLab.data



Explanation: In this case the $3 field is Tiffany’s first month’s contribution and adding the words +10, it adds 10 to $3 which is 15 to make it 25. If Frodo was in the /Frodo/ then it would add 10 to Frodo’s first month or someone else’s and so on.

1. Change Samwise Gamgee's name to Sean Astin The command that we need to enter here is: awk -F: '{gsub(/Samwise Gamgee/,"Sean Astin")} {print $1}' AwkLab.data What this does is use gsub to substitute Sean Astin only for lines that match the pattern match of Samwise Gamgee.

Explanation: What this does is substitute all occurrences of Samwise Gamgee to Sean Astin and then prints the new $1 field which will then be Sean because of the substitution. If Sean Astin was replaced with Gandalf Grey, then every occurrence of Samwise Gamgee would be Gandalf Grey.



1. Write an awk script to do the following (MUST be an awk script not just a bash script or commands on the commandline)
   1. Prints first name of the all the Tooks followed by their total campaign contributions .
   2. Prints "Bullroarer’s contributions after his name
   3. Prints all the names and last month's contribution of those who contributed over $175 for their last contribution

Explanation:

#!/bin/awk -f is what signifies we’ll be running an awk script/file. This works similar to the problems above.

The BEGIN {FS = “:” } is a way to tell Sed to use : as a field separator and to start the file.

Awk then pattern matches for lines that only have Tooks and then prints the 3 month total campaign contributions for each person.

Awk then does the same for b except it just prints Bullroarer’s contributions after his name.

Awk then sets the last month’s contribution to sum and then if sum is greater than 175, it prints the people who had a corresponding contribution on the line.

Sample Run of the Script:

Text

Description automatically generated

Script:

Graphical user interface, text, application, email

Description automatically generated

Script:

#!/bin/awk -f

#a. Prints first name of all the Tooks followed by their total campaign contributions

#awk -F: '/Took/ {print "The Total Campaign Contributions for: " $1 " were: " $3 + $4 + $5}' AwkLab.data

#b. Prints Bullroarers contributions after his name.

#awk -F: '/Bullroarer/ {print "Bullroarers Monthly Contributions were: $" "$"$3 ", " "$"$4 ", and ""$"$5 }' AwkLab.data

#c. Prints all the names and last month's contributions of those who contributed over $175 for their last contribution

#awk -F: '{sum=$5}sum>175{print $1 " - Their last months contribution was: " sum }' AwkLab.data

#Commands that work^

#Actual File Start \/

BEGIN {FS = ":"}

/Took/ {print "Number1 - The Total Campaign Contributions for: " $1 " were: " $3 + $4 + $5}

/Bullroarer/ {print "Number 2 - Bullroarers Monthly Contributions were: $" "$"$3 ", " "$"$4 ", and ""$"$5 }

{sum=$5}sum>175{print "Number 3 - "$1 " - Their last months contribution was: " sum }

Explanation: The BEGIN {FS = “:””} line is a way of telling Awk to start accepting commands. It’s one way to indicate the field to use as a field separator for all of the following subsequent commands. If you were to change the :, the results would be different since the field values are no longer what they were originally set to.

The second line looks for just Tooks and print’s their campaign contributions. The /Took/ pattern matches for just the Took(s) and then prints the $3, $4 and $5 values which are the first, second and third month since we used the semicolon as the field separator. If the : wasn’t there, we’d get different results depending on what was used.

The Number1, Number2, and Number3 is just to tell what line is what.

By replacing Took, Awk with something like Baggins, Awk would then do what it is doing for the Tooks for the Baggins.

Similar to line 2, the /Bullroarer/ line does what it does the same way as the Tooks to Bullroarer. The number 2 is a way to tell what line is when it prints. If Bullroarer was replaced with Frodo, it would search for just Frodo. By enclosing the “$” in “”, awk just prints the $ as is and then with some formatting, makes it appear next to the contributions. $1 means to print the field value of $1 but “$1” would mean to print an actual $1 on the line.

This last line only prints people whose last months contribution was over 175. It sets the last months contribution to sum but it can be anything, and then says sum or the last month is greater than 175. If the last month or the $5 field is greater than 175, it’ll get printed but otherwise it will not. Adding sum in the brackets without any quotes will print the actual value set earlier in the first initial {}.

https://www.gnu.org/software/gawk/manual/html\_node/Executable-Scripts.html