**Awk**

Justice Escalante

Linux Admin

**Printing all the First Names.**

Text

Description automatically generated



In the screenshot above you can see the command ***awk ‘{print $1}’ AwkLab.data***, and what was spit out by the terminal above. For this data to be outputted properly, first we need to understand the delimiter that we are using based on the data provided. In this case we are using spaces as that is the default for awk and in this scenario it works well. In the second screenshot above you can see the data is generally split up by spaces and :’s, now to use this in awk we must provide it with what section we want to print. For this we will be using ***‘{print $1}’***  as we are only pulling the first names, so the first section out of the data. Awk works in sections/columns, so by the provided delimiter it will split your data by that. So section two in the data would be $2, $3 for section 3 etc.

**Printing phone numbers for Tom and Frodo after their names**



This example we are looking to print phone numbers after two names using Awk, this can be achieved by using the command awk -F: ‘/Tom Bombadil/ || /Frodo Baggins/ {print $1, $2}’ AwkLab.data. To make this specific example easier on our end, we will switch the delimiter to : using the -F: operand in awk. This is more convenient to us as the data is displayed as Frodo Baggins:(206) 548-1278:250:80:75 and can be separated a lot better by semi colons. With the sections separated by colors, I will show you the difference between using the default(space) or :. Frodo Baggins:(206) 548-1278:250:80:75(Space) VS Frodo Baggins:(206) 548-1278:250:80:75(:). As you can see the sections would be completely different depending on the delimiter. Now that we have this established, in the data we can see a clear split between the name phone # and the rest of the data. Luckily for us the sections using :’s are just in $1 and $2. Now to get both names to be scanned/searched for these specifics we also must add a or operand such as ||. This will look for Frodo or Tom when doing the search.

**Printing Peregrin’s full name and phone number area code only.**



Using ***awk -F: '/Peregrin/ {print $1, $2}' AwkLab.data | awk '{print $1, $2, $3}'*** we can execute this example. First we need to split the data in smaller sections so we get just the full name and number. To do this we used the delimiter : as the data where looking for is between it. When Using ***awk -F: '/Peregrin/ {print $1, $2}' AwkLab.data*** we are getting back the data as ***Peregrin Took (510) 548-5258***, now with this we can see a clear separation by spaces instead of both spaces and :’s we can section out the data a lot easier then its original. Now that we have the data from the first command like this, we can pipe it back into awk to now use the default delimiter space to grab out said data way more efficiently. Now to pull out just the full name and are code we can run a pipe proceeded by the new awk command ***awk '{print $1, $2, $3}'.*** This will now pull the sections 1-3, as 1-2 is the full name and the 3rd section is the area code that were looking for as well.

**Printing all phone numbers (full number) in the 123 area code along with the names**

Text

Description automatically generated

In this scenario we used the command ***awk -F: '/(123)/ {print $1, $2}' AwkLab.data*** to get the data outputted above, which was the full name and number of people in the 123 area code. In this example, since the specifics to search for was provided to us it makes it a lot easier. We are looking for the (123) area code so right of jump we can provide awk with a regex pattern to search for which lines would contain it, this pattern would be ***/(123)/***. Now that this step is out the way, we can now cut the sections out that we want using awk, using :’s as the delimiter we can see that the first/last name are in the first section of the data, and the second section has the full phone number. Here below is how the data is formatted for better clarification.

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

**Printing all Last names beginning with either a T or D (careful of middle names!)**

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated with medium confidence

In this example since there is middle names, we need a few extra steps to make sure were not grabbing those as the last name but rather true last names. For this example the command ***awk -F: '{print $1}' AwkLab.data | awk '{print $NF}' | awk '/^(T|D)/ {print $0}'*** was used. In this command there are two pipes used to refilter data into awk and to organize it better. The first section of the command is as simple as using the delimiter of : and isolating just the first-last name by itself so then we could go further with specifics (Screenshot above of just this command isolated by itself). Now that we have just full names and nothing else to conflict with, we can proceed to filtering just the last names with T/D. As some of these names have a middle name, we can’t just simply search for T/D as it might grab first last or middle names. Luckily because of the way names are formatted, we know that the last name will always be the last part/section of a name regardless of a middle name or not so we can use this to our advantage. In the second part we piped the full names into a second awk command that simply just pulls the last section off; ***| awk '{print $NF}'***. As the data now would be First Last or First Mid Last we now can use space again as a delimter as well and simply use the operand ***$NF*** to pull the last section as technically we don’t know if the last name falls under section 2 3 or ?, so its easier just to pull the last section off and that’s it. Using ***NF*** which is equal to the number of fields in said line, we can simply just use ***NF*** as a variable when calling upon fields if we are unsure if a middle will add a extra field or not resulting in inaccurate data when calling upon said fields. Having just last names now make it super simple for us as all we have to do is use a regex expression such as ***'/^(T|D)/’*** to match anything that starts with a T or a D. After matching we can simply print all that was matched with awk with ***$0***. This could be put/made into a script for better organization, but for easier documentation purpose we will do it this way.

**Printing all first names containing four or less characters.**



For this problem we can first start by using space as a delimiter as it’s a quick way to separate just first names. Moving on from that, we can now use the command ***‘length($1) <= 4 {print $1}’ AwkLab.data*** to finish off the rest of the problem. Using the function length in this example as ***‘length($1) <= 4*** helps immensely as were simply only looking for names containing four chars or less. ***Length()*** is a great function to use in cases like this as we can pick and choose words with specific lengths. Now that the specifics are done we can simply add the ***{print $1}*** to print that first section containting the first names. This problem can look a bit intense but it isn’t to bad when you understand some of awks operands.

**Printing the first names and area codes of all those in the 916 area code**

Text

Description automatically generated

Text

Description automatically generated

This problem is quite unique as the two pieces of information we need are not attainable by one single delimiter but rather two. For this problem working backwards can help as we wont have to deal with the possibility of middle names conflicting as we count how many sections to get to the area code. Since we know the area code is a one spaced delimiter if we work from right to left we can use this. Using ***NF*** we can do ***${NF-3}*** and ***${NF-2}*** to tell awk to go backwards and first pull the 3rd field and then the 2nd field as we don’t need the last field containing the remaining bits of the number but have to find a way to exclude it.

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



This one is rather simple as were are given specifics on what to search for, above the command ***awk -F: '/Sacharissa Cripslock/ {print $1, "$"$3, "$"$4, "$"$5}' AwkLab.data*** was used to find the output above. Since we know were looking for Sacharissa’s contributions specifically, we can add that in the regex pattern to search for so only that line comes up when using awk. Now to print the contributions this is as quite simple as listing the fields after each other. In this case fields 3,4,5 are what she had contributed. Since were only referencing the number and no dollar amount here, we have to add ***“$”*** in front of each field so we could display that properly.

**Printing last names followed by a comma and the phone number. Be careful of the last names’ format.**

Text

Description automatically generated

Using one of the tricks from the previous prompt, we can use the command ***awk -F: '{print $1, $2}' AwkLab.data | awk '{print $(NF-2)",", $(NF-1), $NF}'*** to find the last names followed by the numbers. Now there’s plenty of ways you can do this, but since I already used a command previously that could help in this, I did. Once again in this problem I have utilized the NF operand to print the fields going right to left instead of the normal order. First I used ***awk -F: '{print $1, $2}' AwkLab.data*** to just pipe the full name and number by them selves to make it easier. Now that the data looks like ***Samuel Vimes (510) 548-1278***, we can use this beautiful backwards feature to print the last field containing using $NF, then the second to last field (510) using ***$(NF-2),*** and lastly just the last name using ***$(NF-2)***. But we have to include a comma as well which could be done with a “,” put as needed inside of the print portion, so it prints it as needed.

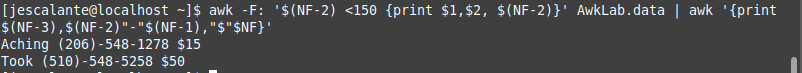
**Printing the first and last names of those who contributed more than $110 in the last month. Make sure to include their last month contribution amount after the name**

Text

Description automatically generated

Using ***:***’s as a delimiter, we can find the proper output of Samuel Vimes $175 as one instance by using the command ***awk -F: '$NF > 110 {print $1,"$" $NF}' AwkLab.data***. As hard as this one ***looks***, it can be rather quick as the main specifics if names are being printed are if the last month(last section) is above 110, Knowing this, we can give awk a rule to follow by, in this case we used ***$NF > 110*** as the last field == the month and were only looking for names above 110. If this criterion is met, awk can continue with the rest of the command which would be pretty the first delimited section(names) and then proceeded by the last contribution formatted with a ***$*** in front of it.

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



For this example we used ***NF***/the number of fields again as we are dealing with only the last name. If we knew for sure were only had first names and last names we wouldn’t have to do this. But since we are unsure, we will set it up this way. First we are looking for those who contributed less then 150$ in the first month, so based on the data the first month of contribution is setup right after the phone number or 3 sections going from the back. Because we know this we can simply use the NF operand and subtract 2 from it so we get the field that is 3rd to last which would be the first month. Now that we have the field, we can have it filter out anyone who contributed less then 150 with the command ***awk -F: '$(NF-2) <150 {print $1,$2, $(NF-2)}' AwkLab.data***. That command above with look for the people who did less then 150 per that month and return their name number and the contribution in question(screenshot above). Pretty much a setup for the next command we will pipe. The next command we pipe is ***awk '{print $(NF-3),$(NF-2)"-"$(NF-1),"$"$NF}',*** this will simply take the first set of piped data and re configure it using NF once again to display only the last name and other specific information. For better documentation/looks, I’ve added ***“$”*** and “-“ in the print portion of the command so that we can see the formatting better.

**Printing the first names and contribution of those who contributed between $10 and $200 in the first month.**

A screenshot of a computer

Description automatically generated



This example is very similar to the one above but with slight iterastions. To pull of this example the command ***awk -F: '$(NF-2) < 200 && $(NF-2) > 10 {print $1, $(NF-2)}' AwkLab.data | awk '{print $1, “$”$NF}'*** was used. First we found the first month using ***$(NF-2) < 200 && $(NF-2) > 10*** to see who contributed between 10 and 200, this gave us full names and just the last contribution like shown above. Now we can repipe this into awk to get rid of the potential mid/last names by only referencing the first section and the last NF as we know the last and middle name wont be there. This was done with the following pipe of awk ***'{print $1, "$"$NF}'\***

**Printing the first name, last names and total contributions of those who contributed less than $700 over the three-month period.**

Text

Description automatically generated

Text

Description automatically generated

In this example we used the command ***awk -F: '{if (($4+$5+$6)<700) print $1,($4+$5+$6)}' AwkLab.data | awk '{print $1, $(NF-1), "$"$NF}’*** to find out of those who contributed less then 700. The first section of the command ***awk -F: '{if (($4+$5+$6)<700) print $1,($4+$5+$6)}' AwkLab.data*** is used primarily for the contribution filtering. It uses if statements to add all 3 last months/sections of contributions to see if they total less then 700. If they have reach this criteria it will print the full name of the person alongside the total number which can be seen in the print portion by all three sections 4,5,6 being added up together. The second portion ***awk '{print $1, $(NF-1), "$"$NF}*** simply takes the full name and only returns you with the first and last name only proceeded by the total with an added ***“$”***. Since we don’t know if the middle name is present, we can use the Number of fields minus 1 as the last name piped is only the second to last field.

**Printing the first names and first letter of the last name, and average contribution of those who had an average contribution of more then $300**

Text

Description automatically generated

***awk -F: '{if (($4+$5+$6/3)>300) print $1,(($4+$5+$6)/3)}' AwkLab.data | awk '{print $1, substr($(NF-1),1,1), "$"$NF}'*** was the command used above to get the corresponding data. This example is extremely similar to the previous one, but this time were averaging out, and then also cutting the string of the last name to just the first letter. The first part of the command that contains ***awk -F: '{if (($4+$5+$6/3)>300) print $1,(($4+$5+$6)/3)}' AwkLab.data*** will search using if, it will check if the last 3 months(fields 4,5,6) are greater than 300 and then will proceed to print the full name followed by the average. Now the final command that has the full name and average will be cut up using ***substr(*** to provide only the first letter of the last name field that is found at the location of ***$NF-1.***

**Printing the last name and area code of those not in the 916 area code**

Text

Description automatically generated

Taking what we have learned already, to start this one off we have first used a rgex pattern to look for the inverse/opposite of the (916) area code by using the part of the command ***'!/(916)/,*** the ***!*** before makes it from looking for 916, to the inverse of 916. Now that we have all the people that don’t have the 916 area code, we can proceed to print just the last name and their designated area. Since we used space as the delimiter, luckily enough the last name and area code are attached as follows: ***Justice:(978)*** so we can do a simple call to just that field. Since there could be middle names, we will use NF once again for this, now that we have the piece of info were looking for all we have to do is simply replace the : with a ***gsub*** function before calling the print to be done. This was all done in this example by the command ***awk '!/(916)/ {gsub(/:/, " ", $(NF-2)); print $(NF-2)}' AwkLab.data.*** Above you can see ***{gsub(/:/,” “, being used on $(Nf-2)*** as this is were the last name sits and we want to remove the :.

**Printing each record preceded by the number of the record.**

Text

Description automatically generated

For this example we can use the ***NR*** variable as it represents the number of records. The above command that was executed was ***awk -F: '{print NR":"$0}' AwkLab.data,*** pretty much were printing all the data and associated its respective number with it.

**Printing the name and total contribution of each person.**

Text

Description automatically generated

In this example we need the total contribution of each person, as we know the total is all the fields with contributions added together. To achieve this I have used the command ***awk -F: '{print $1,"$"($4+$5+$6)}' AwkLab.data***, its as simple as printing the first field containing the names(using the delimiter of :) and then proceeded by the fields all added together to get the total.

**Adding $10 to Tiffany Aching’s first contribution and print her full name and first contribution.**



Bouncing off what we learned above, this one is quite similar to adding all the contributions, but instead of adding multiple fields we will be simply added 10$to one. Using the command ***awk -F: '/Tiffany Aching/ {print $1,($3+10)}' AwkLab.data*** we can execute this, first we print out the name using ***$1*** and then the first contribution with an addition of 10 by calling for it in parenthesis with ***$3***. We also have to search specifically for Tiffany Aching, but with the use of a regex expression this makes it super easy as all we do Is just supply the pattern with the name we need.

**Changing Samwise Gamgee’s name to Sean Astin**



For the problem above we used the command ***awk -F: '/Samwise Gamgee/ {$1="Sean Astin"; print $0}' AwkLab.data*** to change the name in the field. To do this we can simply set up the field to be reprinted differently, using ***$1="Sean Astin";*** we can complete this as we are giving the first field that contains the name a new value of Sean Astin. Now that we have set a new value in the awk command, when we go to print all using ***$0*** it will be updated with the new name we gave it. Don’t forget that in this scenario we also have to use the original name once again as the regex pattern to specifically look for that name to be replaced.

**Write an awk script to do the following (MUST be an awk script not just a bash script or commands on the commandline) (a) Prints first name of the all the Tooks followed by their total campaign contributions . (b) Print the full names and contributions of anyone who contributed between $10 and $200 in the last contribution (c) Prints the full names and average contribution of those who contributed less than $300 on average**

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

As complicated as it might seem to make an awk script, it’s not to bad when you think of it as multiple awk commands just being executed at once instead of one by one. For this awk script we had 3 main commands that were ran that did numerous things. To initiate the script we need a schbang to tell the system what to do: #!/usr/bin/awk -f, we also told awk what delimiter to use with the BEGIN{ and setting FS to :. The first part of the awk script /***Took/ {print $1, "$"($4+$5+$3) |'{print $1,$NF}'"*** was made to find all the Tooks and there total campaign contributions. First we need to use a regex pattern/expression to look for the data that is only represented by the looks. After that we can proceed to add the totals and print the name. Since we know fields 3,4,5 are for the contributions, we can add then all together to be printed alongside the name. Since the delimiter of : was used originally we would have had full names, but since we only want the first names we could repipe this and have the total = last field/***NF*** and then just the first name with ***$1.*** Now the second portion of the script has if statements as we are looking for a range. To do this we simply put the if statements first as shown: ***$5 > 10 && $5 < 200 {print $1,"$"$5,*** proceeded by what will be printed if that was hit. Since we are looking at the final contribution, we can reference ***$5*** for this and print the full name and $5 once again if that criterion is met. Lastly, we have ***(($4+$5+$3)/3) < 300 {print $1,"$"(($4+$5+$3)/3)*** which is very similar to what we just did, but adding a, little extra flavor, This one is a if statement as well but now were are looking at if the average is < 300. So we would also have to add all 3 fields and average them out. If the < 300 if statement is met for the last 3 fields then can print the full name using the first field $1, proceeded by the average that we calculated to do that.

**Bibliography**

<https://www.youtube.com/watch?v=9YOZmI-zWok>

<https://www.math.utah.edu/docs/info/gawk_8.html>

<https://linuxhint.com/awk-nf-ubuntu/>

<https://linuxhint.com/nr-built-in-awk/>

<https://www.freecodecamp.org/news/how-to-exit-vim/>

<https://unix.stackexchange.com/questions/631662/count-the-number-of-field-by-using-awk-only-and-without-other-commands-such-as-u>

<https://stackoverflow.com/questions/9841036/awk-count-number-of-fields-and-add-accordingly>

<https://askubuntu.com/questions/1064895/how-to-use-gsub-function-awk-to-replace-characters-with-empty-value>

<https://www.math.utah.edu/docs/info/gawk_8.html>

<https://www.youtube.com/watch?v=oPEnvuj9QrI>

<https://www.youtube.com/watch?v=fCw-xf31M_s>