**Grep**

Justice Escalante

Linux Admin

**Printing lines containing the string Lane**

Text

Description automatically generated

In this case using ***grep Lane GrepLab*** works just fine for finding “Lane” in the file as no other specifics are needed beside just Lane, so no operands will be needed alongside grep. If you wanted it to find Lane by itself and not CarltonLane like shown in the screenshot above you could use -w to match just whole words

Text

Description automatically generated

**Printing where the person’s first name starts with H**

Text

Description automatically generated

Luckily for us in this scenario the First names are up front so it makes it a tad bit easier in terms of what operands to use with Grep. So for this one I used ***grep ^H GrepLab****,* using ^ as an operand in this command works great as it searches the start of the line. So ***^H*** would be searching for any line whose first name starts with a capital H, if lets say the names weren’t capitalizes or some weren’t, you could use -i to desensitize the command to caps or not.

**Printing all lines ending in three zeros (000)**

Text

Description automatically generated

To use grep in terms of the endings in what your searching, $ would be the operand you would use as its specific for endings. Above you can see we have an example where multiple lines end in 000 when the command is called upon, using ***grep 000$ GrepLab*** this is possible. Below you can see I redid the command but with no ending operand and only two zeros for demonstration purposes ***grep 00 GrepLab***

Text

Description automatically generated

**Printing all lines that don’t contain 408**

Text

Description automatically generated

Searching for any given specific value using Grep is practically the same as looking for what values/lines do not contain that value. Using grep, there is a operand ***-v*** that takes whatever you support it with, and returns you the inverse. Above you can see I did just that as were looking for lines that Do Not contain 408, so I used the command ***grep -v 408 GrepLab*** to achieve that as shown. I will also include a image showing the opposite below.

Text

Description automatically generated

**Printing lines where birthdays are in the year 1935 (be careful of the date format! it’s MM/DD/YY)**

Text

Description automatically generated

Knowing that the date format is in MM/DD/YY we can use grep to isolate just the year, so in this case I used ***grep /35 GrepLab*** as were looking for the birthdays in the year 1935 which is displayed by ??/??/35. You could also use the date format if your trying to be more specific, but in this example /35 works great.

**Printing all lines where the phone number is in an area code that starts with an 8**

Text

Description automatically generated

In this case it gets a little more tricky as what we are looking for is starting to be more specific. As we are looking for area codes on phone numbers that start with 8, we can use expressions which are almost like patterns that you give the computer to exclude or look for certain properties. In the screenshot above I used ***grep ‘:8..-‘ GrepLab***, since we know it starts with 8 and the data is formatted with a : before the number we can use that. So with *:8* so far, we can add ***..*** which is regular expression symbol that means any character can fill that spot when searched upon. If a number was 897 or 865 it would work as we have the 8 and two random numbers following, I have also added a ***–*** at the end since typically the area code format for a number ends after 3 numbers so it could help limit some inaccuracies when searching.

**Printing all lines containing an uppercase letter, followed by 4 lowercase letters, a space and one uppercase letter**

Text

Description automatically generated

Using grep on this one gets more involved, now regex/regular expressions are a lot more helpful when it comes to problems/searches like this as it is a lot more specific. For this one I have used ***grep ‘[A-Z][a-z]\{4\} [A-Z]’.*** This expression looks complicated but is in fact quite simple, first you have the beginning part [A-Z] which means any string with a capital letter A-Z will be looked at. Secondly we have [a-z]\{4\} which is almost the same, lowercase letters [a-z] as requested and 4 of them which gives the operator \{4\}. This meaning [a-z] is looked for 4 times in a row as we want the capital letter followed by 4 lower case. You could do [a-z][a-z][a-z][a-z] as well, but when things get more complicated you want the command line/code to look less clustered. What if you needed 65 lower case letters?!? Lastly its followed by a space and the remaining capital letter part to match the pattern which would be [A-Z].

**Printing lines where the address begins with a two or three digit number (so this would be 12 main st or 123 main street but not 1234 main street).**

A screenshot of a computer

Description automatically generated with medium confidence

In the screenshot above you can see the command ***grep -E “:[0-9]{2,3} “ GrepLab*** has pulled addresses with only two or three digit numbers with the power of an expression. Using the beginning operand :[0-9] we can isolate our search to just numbers after the :. With {2,3} and a space after we can further make our search specific as now we are looking for a number 0-9 after the : and only in cases where the length of the address is 2-3 digits followed by a space. So if it was 4 digits it wouldn’t work as the 4 digit would be filled and not be a space making the search invalid.

**Printing lines where the person lives in Mass or Illinois**

Text

Description automatically generated

For this one its quite simple, as where only looking for MA or IL, we can make an expression that looks for one or the other. To find this specific result, the command used was ***grep ‘MA\|IL’ GrepLab***. The use of \| in this command is as simple as a OR operand, since were looking for MA or IL, those where the values inputted in the pattern

**Printing lines containing the addresses that aren’t on a street (You might see St as shorthand for street)**

Text

Description automatically generated

For this example, I used a OR operand as well as the two iterations of how street was used. Since were looking for addresses that aren’t on a street, we can search for ones that are on a street and then use the inverse operand to find what were looking for. In cases like this, sometimes its easier to look for something specific and then use its inverse. As we know the streets are written with either St., or Street, this gave us perfect specifics to find the ones that don’t contain it with -v. Down below I ran ***grep -v ‘St.,\|Street,’ GrepLab*** and it gave me the addresses without street in them as we used the inverse operand -v.

Text

Description automatically generated

**Bibliography**

[**https://docstore.mik.ua/orelly/unix/unixnut/ch06\_04.htm**](https://docstore.mik.ua/orelly/unix/unixnut/ch06_04.htm)

[**https://www.geeksforgeeks.org/how-to-use-regular-expressions-regex-on-linux/**](https://www.geeksforgeeks.org/how-to-use-regular-expressions-regex-on-linux/)