## More on Grep

1. Print all the lines having the word "pattern".

```grep -w "pattern" text.txt```

2. Pick out the blank lines in the file

```grep "^$" text.txt```

3. Count total number of empty lines in the file.

```grep "^$" text.txt|wc -l```

4. Print the line which have both “Sir and Madam”.

```grep -e "Sir" -e "Madam" text.txt```

5. pick out lines with “pattern1” “pattern2” or “pattern3”. (use the alternator |)

```egrep "hello|pattern|line" text.txt```

6. pick out lines that have at least two p's followed by any number of letters followed by 'ore'. The p's do not have to be next to each other.

```grep -E 'p.\*p.\*ore' text.txt```

7. pick out all the lines with v, z or I in them

```egrep "v|z|I" text.txt```

8. pick out all the lines that do not start with an uppercase letter.

```code```

9. pick out all the lines that end with a dash -

```grep "\-$" text.txt```

10. pick out all the words that end with ore

```grep -o '\b\w\*ore\b' text.txt```

11. pick out all the words that start with f or F

```grep -o '\b[Ff]\w\*' text.txt```

12. pick out lines that uses first letter alliteration - starting two words with the same letter.

```egrep '(\b\w+\b).\* \1' text.txt```

13. determine how many times contains the word "pattern".

```grep -o "pattern" text.txt | wc -l```

14. to pick out lines with at least 40 characters:

```grep '.\{40\}' text.txt```

15. to pick out lines with no punctuation

```grep '^[A-Za-z0-9 ]\*$' text.txt```

16. to pick out lines with an uppercase letter other than the first character. (The first character on the line does not count.)

```grep '[A-Z].\*[A-Z]' text.txt```

17. To pick out lines without rav

```grep -v "rav" text.txt```

# 1. print lines numbers of lines beginning with "O"

`sed -n '/^o/=' a.txt`

# 2. delete digits in the given input file.

`sed 's/[0-9]//g' input.txt`

# 3. delete lines that contain both BEGIN and END

`sed '/BEGIN.\*END/d' input.txt`

# 4. delete lines that contain BEGIN but not END

`sed '/BEGIN/!b;/END/!d' input.txt`

# 5. deletes the first character in each line in a file

`sed 's/^.//' input.txt`

# 6. deletes the last character in each line in a file

`sed 's/.$//g' input.txt`

# 7. deletes the first character, (if it numeric ) in each line in a file.

`sed 's/^[0-9]//' input.txt`

# 8. deletes the character before the last character in each line in a file.

`sed 's/.\(.\)$/\1/' input.txt`

# 9. that swaps the first and second words in each line in a file.

`sed 's/\([^ ]\* \)\([^ ]\*\)/\2 \1/' input.txt`

# 10. replace the word "mcis" with "sois" in the first five lines in a file

`sed '1,5s/mcis/sois/g' input.txt`

# 11. add "SOIS" prefix to all the lines

`sed 's/^/SOIS /' input.txt`

# 12. add "." at the end of each line in the file.

`sed 's/$/./' input.txt`

# 13. Pick the line with Social security number in the format of 999-99-9999

`sed -n '/^[0-9]\{3\}-[0-9]\{2\}-[0-9]\{4\}$/p' input.txt`

# 14. Pick the Valid IP address of the computer (4 numbers separated by '.'). (e.g. 192.168.0.1)

`sed -n '/^[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}$/p' input.txt`

# 15. Pick the Valid URL beginning with "http://". (e.g. http://manipal.edu)

`sed -n '/^http:\/\//p' input.txt`

# 16. Pick the Valid email address, assuming 'a-z','0-9','-','.' are the valid characters for user ID, and domain name has to end with either ".com" or ".net"

`sed -n '/^[a-z0-9.-]\*@[a-z0-9.-]\*\.\(com\|net\)$/p' input.txt`

# 17. Print all lines containing words that start with "f" and end with "s".

`sed -n '/\bf[a-z]\*s\b/p' input.txt`

# 18. Print lines containing a capital "L", but not as the first character on the line.

`sed -n '/^.[^L]\*L/p' input.txt`

# 19. matches dates in the American MM-DD-YYYY format where months and days can be 1 or 2 digits, years must be 4 digits starting with a 1 or a 2, and the delimiter is either "-" or "/" but not both.

`sed -n '/^\(0\?[1-9]\|1[0-2]\)\([/-]\)\(0\?[1-9]\|[12][0-9]\|3[01]\)\2\(1\|2\)[0-9]\{3\}$/p' input.txt`

# 20. print all the lines of a file that are less than 10 characters in length.

`sed -n '/^.\{1,9\}$/p' input.txt`

# 21. replace all occurrences in a file of "the" with "a" and "The" with "A".

`sed 's/\bthe\b/a/g; s/\bThe\b/A/g' input.txt`

# 22. substitute the word "button" for "tree", only if tree occurs at the end of a line.

`sed 's/tree$/button/' input.txt`

# 23. substitute the word foible for the word tree but not the word trees.

`sed 's/\btree\b/foible/g' input.txt`

# 24. change a text so that every period at the end of a sentence is changed to an exclamation point (!) and every question mark is replaced with an ellipsis (...)

`sed 's/\.\([[:space:]]\|$\)/!\1/g; s/?/.../g' input.txt`

# 25. Change every occurrence of the word 'me' with 'you' but only at the end of a line.

`sed 's/\bme$/you/' input.txt`

# 26. change a document so that every occurrence of a three-letter abbreviation for a month is replaced by the appropriate number, i.e., Jan is replace by 1, Feb by 2 etc.

`sed 's/\bJan\b/1/g; s/\bFeb\b/2/g; s/\bMar\b/3/g; s/\bApr\b/4/g; s/\bMay\b/5/g; s/\bJun\b/6/g; s/\bJul\b/7/g; s/\bAug\b/8/g; s/\bSep\b/9/g; s/\bOct\b/10/g; s/\bNov\b/11/g; s/\bDec\b/12/g' input.txt`

# 27. match lines containing (i) Jefferies jeffery jeffeys (ii) hitchen hitchin hitching (iii) Heard herd Hird (iv) dix dicks dickson dixon (v) Mcgee mcghee magee.

`sed -n '/\b[Jj]effer(ies|y|ys)\b\|\b[Hh]itch(en|in|ing)\b\|\b[Hh](eard|erd|ird)\b\|\b[Dd]i(x|cks|ckson|xon)\b\|\b[Mm]c?g[he]{1,2}\b/p' input.txt`

# 28. replace all multiple spaces in a string by only one space.

`sed 's/ \*/ /g' input.txt`

# 29. A line comment in C is introduced by the sequence //. Alternatively the comment can be enclosed by /\* and \*/. Write a sed command which transforms the // comments to the enclosed ones.

`sed 's|//\(.\*\)$|/\*\1\*/|' input.txt`

# 30. Print alternative lines (line1,3,5…)

`sed -n '1~2p' input.txt`

# 31. Change the letters "dog" to "HORSE" everywhere it occurs on all lines.

`sed 's/dog/HORSE/g' input.txt`

# 32. Change all occurrences of the letters "Man" at the beginning of a line to "Person".

`sed 's/^Man/Person/' input.txt`

# 33. Change all occurrences of "stick" followed by any punctuation at the end of a line to "Stick.". (The punctuation is replaced by a period.)

`sed 's/stick[[:punct:]]$/Stick./' input.txt`

# 34. Change all occurrences of "Dog" or "dog" to "COW".

`sed 's/[Dd]og/COW/g' input.txt`

# 35. Change all Canadian or American spellings of colour (color) to "Color".

`sed 's/colou\?r/Color/g' input.txt`

# 36. Double all vowels in every word on every line.

`sed 's/\([aeiou]\)/\1\1/g' input.txt`

# 37. Triple the amount of space between every word.

`sed 's/ / /g' input.txt`

# 38. Find and print lines that contain "dog" followed by any number of digits then "cat".

`sed -n '/dog[0-9]\*cat/p' input.txt`

# 39. Find and print lines that contain the letters "dog" followed anywhere by the letters "cat".

`sed -n '/dog.\*cat/p' input.txt`

# 40. Change all occurrences of one or more digits to the single word "NUMBER".

`sed 's/[0-9]\+/NUMBER/g' input.txt`

# 41. Replace all occurrences of one or more blanks with a single blank.

`sed 's/ \*/ /g' input.txt`

# 42. Replace all occurrences of one or more tabs or blanks with a single blank.

`sed 's/[[:space:]]\+/ /g' input.txt`

# 43. Remove the first 8 characters from every line.

`sed 's/^.\{8\}//' input.txt`

# 44. Remove all leading blanks or tabs from all lines.

`sed 's/^[[:space:]]\*//' input.txt`

# 45. Remove all trailing blanks or tabs from all lines.

`sed 's/[[:space:]]\*$//' input.txt`

# 46. Replace all tab characters with eight spaces.

`sed 's/\t/ /g' input.txt`

# 47. Change all punctuation so that the sentence period lies outside of the closing double quote, e.g. "Hello there." becomes "Hello there".

`sed 's/\([^"]\)\."$/"\1./' input.txt`

# 48. Remove everything leading up to and including the last blank on each line.

`sed 's/.\*[[:space:]]//' input.txt`

# 49. Remove everything including and after the first blank on each line.

`sed 's/[[:space:]].\*$//' input.txt`

# 50. Put double quotes around every occurrence of the phrase "user-friendly".

`sed 's/user-friendly/"user-friendly"/g' input.txt`

# 51. Add an extra blank after every period at the end of a sentence.

`sed 's/\.\([[:space:]]\|$\)/. \1/g' input.txt`

# 52. Make sure that every period at the end of a sentence is followed by exactly two blanks.

`sed 's/\.\([[:space:]]\{0,1\}\)/\. /g' input.txt`

# 53. Truncate every line to ten characters.

`sed 's/^\(.\{10\}\).\*/\1/' input.txt`

# 54. Exchange the first 10 characters with the next 15 characters on every line.

`sed 's/^\(.\{10\}\)\(.\{15\}\)/\2\1/' input.txt`

# 55. Exchange the first number with the second number on every line.

`sed 's/\([0-9]\+\).\*\([0-9]\+\)/\2\1/' input.txt`

# 56. Remove all leading zeroes from the first number on each line. Don't mishandle single digit zeroes.

`sed 's/^0\*\([1-9][0-9]\*\|0\)/\1/' input.txt`

# 57. Find and print lines that contain all the vowels in alphabetical order, a before e before i before o before u. Test using /usr/dict/words.

`sed -n '/a.\*e.\*i.\*o.\*u/p' /usr/dict/words`

# 58. Find and print lines that contain all the vowels in any order. Test using /usr/dict/words.

`sed -n '/a/p; /e/p; /i/p; /o/p; /u/p' /usr/dict/words | sort | uniq -c | sed -n '/5/p' | sed 's/^[[:space:]]\*5[[:space:]]\*//'`

# 59. Change all occurrences of one or more digits surrounded by spaces to the word "NUMBER" also surrounded by spaces.

`sed 's/[[:space:]][0-9]\+[[:space:]]/ NUMBER /g' input.txt`

# 60. Change only the second occurrence of a single blank to a colon in each line.

`sed 's/ /:/2' input.txt`

# 61. Change the only the second-to-last single blank to a colon in each line.

`sed 's/ \([^ ]\*\)$/:\1/' input.txt`

# 62. Change only the second occurrence of a string of one or more blanks to a colon in each line.

`sed 's/[[:space:]]\+/:/2' input.txt`

# 63. Change only the second-to-last occurrence of a string of one or more blanks to a colon in each line.

`sed 's/\(.\*\)[[:space:]]\+\([^[:space:]]\+[[:space:]]\+[^[:space:]]\+\)$/\1:\2/' input.txt`

# 64. Remove all occurrences of HTML tags whose open and closing angle brackets are on the same line (e.g. \<BR>, \<TABLE>, \<A HREF="...">, etc.). Remove all of them, not just the first ones.

`sed 's/<[^>]\*>//g' input.txt`

# 65. Remove everything on every line that appears between double quotes, leaving only the quotes. (Example: a "bcd" efg "h i" j --> a "" efg "" j ) Handle empty strings (adjacent quotes) correctly

`sed 's/"[^"]\*"/""/g' input.txt`

# 66. Find lines that contain only one single quote character (an unmatched quote).

`sed -n '/^[^'"'"']\*'"'"'[^'"'"']\*$/p' input.txt`

# 67. Put double quotes around every occurrence of the phrase "user-friendly", unless the phrase already has double quotes around it.

`sed 's/\b\(user-friendly\)\b/"\1"/g; s/""user-friendly""/user-friendly/g' input.txt`

# 68. Find all numbers prefixed by a dollar sign, remove the dollar sign, and suffix the number with "CDN", e.g. $123.45 becomes 123.45CDN. Now do the reverse.

`sed 's/\$\([0-9.]\+\)/\1CDN/g' input.txt`

`sed 's/\([0-9.]\+\)CDN/\$\1/g' input.txt`

# 69. Find all numbers with periods separating decimals and change the periods to commas, e.g. 123.45 becomes 123,45. Now do the reverse.

`sed 's/\([0-9]\+\)\.\([0-9]\+\)/\1,\2/g' input.txt`

`sed 's/\([0-9]\+\),\([0-9]\+\)/\1.\2/g' input.txt`

# 70. Find all numbers with commas separating sets of three digits and change all the commas to spaces, e.g. 1,234,567.23 becomes 1 234 567.23. (You may assume the only use of a comma immediately followed by three digits is as a separator.)

`sed 's/\([0-9]\+\),\([0-9]\{3\}\)/\1 \2/g' input.txt`

# 71. Locate common misspellings and mistypings of "@college.com" and fix them all. (e.g. fix manipal.edu, etc.)

`sed 's/@[a-zA-Z]\*\.\(com\|edu\)/@college.com/g' input.txt`

# 72. Remove either single or double quotes from around all strings of one or more digits, e.g. "10" or '10' become just 10. Now do the reverse (add quotes to all numbers).

`sed 's/['"'"'"]\([0-9]\+\)['"'"'"]/\1/g' input.txt`

`sed 's/\b\([0-9]\+\)\b/"\1"/g' input.txt`

# 73. Locate hexadecimal numbers having the form "0xA0FF2375C3" and prefix them with the string "(HEX:)", e.g. 0xDEAD would appear as (HEX:)0xDEAD and 0xBEAD00BEAD00 would appear as (HEX:)0xBEAD00BEAD00. Now do the reverse (remove the prefixes).

`sed 's/\b0x[0-9A-Fa-f]\+\b/(HEX:)&/g' input.txt`

`sed 's/(HEX:)\(0x[0-9A-Fa-f]\+\)/\1/g' input.txt`

# 74. Use a single regular expression to change every occurrence of the word "dog" to be "dog-eat-dog" and "cat" to be "cat-eat-cat". Now do the reverse.

`sed 's/\b\(dog\|cat\)\b/\1-eat-\1/g' input.txt`

`sed 's/\b\(dog\|cat\)-eat-\1\b/\1/g' input.txt`

# 75. Produce a plain list of mail addresses and home pages for everyone with an account on this system.

`sed -n '/^[^:]\*:[^:]\*:[^:]\*:[^:]\*:[^:]\*:\([^:]\*\):\([^:]\*\)/s//\1 \2/p' /etc/passwd`

# 76. Write a script that will perform a simple substitution on the contents of each of the files given on the command line, e.g. $ ./script 's/dog/cat/g' \*.txt

```

#!/bin/

sed\_command="$1"

shift

for file in "$@"

do

sed "$sed\_command" "$file" > "$file.tmp" && mv "$file.tmp" "$file"

done

```

# 77. Have every new sentence in a document start at the beginning of a line. (Insert newline characters at the end of every sentence.)

`sed 's/\.\s/&\n/g' input.txt`

# 78. Find and print lines where all vowels are in strict alphabetical order, i.e. no e precedes an a, no i precedes an e, no o precedes an i, etc. All vowels that appear are in alphabetical order in the input, from left to right. Test your expression on /usr/dict/words.

`sed -n '/^[^aeiou]\*a\*[^aeiou]\*e\*[^aeiou]\*i\*[^aeiou]\*o\*[^aeiou]\*u\*[^aeiou]\*$/p' /usr/dict/words`

# 79. Change the second and all subsequent occurrences of one or more blanks to single blanks. (The first occurrence of a string of blanks is untouched.)

`sed 's/ \*/ /2g' input.txt`

# 80. A file has a large number of columns of numbers separated by blanks. Change every second string of blanks to a colon. (A line of output might appear thus: 12 34:56 78:90 12) You don't know how many columns are in the input files.

`sed 's/ /:/2g' input.txt`

# 81. Exchange the first number with the last number on every line.

`sed 's/\([0-9]\+\).\*\([0-9]\+\)$/\2\1/' input.txt`

# 82. Remove all leading zeroes from all numbers on each line. Don't mishandle a single digit zero.

`sed 's/\<0\*\([1-9][0-9]\*\|0\)\>/\1/g' input.txt`

# 83. would take all the file names in the current directory that end in ".txt" and rename them to end in ".dat". What pattern would you use to rename files with names such as "file1.day.mon" to "file1.mon.day", e.g. "foo.31.01" would become "foo.01.31" and "bar.30.12" would become "bar.12.30", etc.?

`for f in \*.day.mon; do mv "$f" "$(echo "$f" | sed 's/\(.\*\)\.\(.\*\)\.\(.\*\)/\1.\3.\2/')"; done`

# 84. Write a script that will convert alphabetic dated file names to numeric names, e.g. a file named "Mar.12.99" would be renamed "1999-03-12" and "Jul.31.54" would become "1954-07-31". Make sure your script doesn't overwrite any existing files. Does your script handle all the possible forms of each month name, e.g. "Mar", "MAR", "mar", "March", "MARCH", "march"? (Hint: You can use multiple "-e" options to sed.) (p.s. Does your script handle the year 2000?)

```

#!/bin/

for file in \*.\*.\*; do

newname=$(echo "$file" | sed -e 's/\([Jj]an\|[Jj]anuary\|JAN\|JANUARY\)/01/' \

-e 's/\([Ff]eb\|[Ff]ebruary\|FEB\|FEBRUARY\)/02/' \

-e 's/\([Mm]ar\|[Mm]arch\|MAR\|MARCH\)/03/' \

-e 's/\([Aa]pr\|[Aa]pril\|APR\|APRIL\)/04/' \

-e 's/\([Mm]ay\|MAY\)/05/' \

-e 's/\([Jj]un\|[Jj]une\|JUN\|JUNE\)/06/' \

-e 's/\([Jj]ul\|[Jj]uly\|JUL\|JULY\)/07/' \

-e 's/\([Aa]ug\|[Aa]ugust\|AUG\|AUGUST\)/08/' \

-e 's/\([Ss]ep\|[Ss]eptember\|SEP\|SEPTEMBER\)/09/' \

-e 's/\([Oo]ct\|[Oo]ctober\|OCT\|OCTOBER\)/10/' \

-e 's/\([Nn]ov\|[Nn]ovember\|NOV\|NOVEMBER\)/11/' \

-e 's/\([Dd]ec\|[Dd]ecember\|DEC\|DECEMBER\)/12/' \

-e 's/\([0-9]\{2\}\)\.\([0-9]\{2\}\)\.\([0-9]\{2\}\)/20\3-\1-\2/' \

-e 's/\(19[0-9]\{2\}\|20[0-9]\{2\}\)-\([0-9]\{2\}\)-\([0-9]\{2\}\)/\1-\2-\3/')

if [ ! -e "$newname" ]; then

mv "$file" "$newname"

else

echo "Error: $newname already exists. Skipping $file."

fi

done

```

This script handles all forms of month names, years from 1900 to 2099, and avoids overwriting existing files.

 **Count how many packets use each protocol (TCP, UDP, ICMP):**

awk -F"," '{count[$4]++} END {for (protocol in count) print protocol, count[protocol]}' k.h

 **Filter and print only the dropped packets:**

awk -F"," '$8 == "Dropped" {print}' k.h

 **Print Timestamp, Source\_IP, Destination\_IP, and Packet\_Size for packets >1000 bytes:**

awk -F"," '$7 > 1000 {print $1, $2, $3, $7}' k.h

 **Display traffic directed to destination port 443:**

awk -F"," '$6 == 443 {print}' k.h

 **Print all unique Source\_IP addresses:**

awk -F"," '!seen[$2]++ {print $2}' k.h

 **Filter only TCP traffic and calculate the average packet size:**

awk -F"," '$4 == "TCP" {total+=$7; count++} END {if (count > 0) print "Average Packet Size:", total/count}' k.h

 **Count invalid records:**

awk -F"," 'NF != 8 {invalid++} END {print "Invalid Records:", invalid+0}' k.h

 **Extract rows where Source\_IP is in the 192.168.x.x range:**

awk -F"," '$2 ~ /^192\.168\./ {print}' k.h

 **Match traffic directed to either port 80 (HTTP) or port 443 (HTTPS):**

awk -F"," '$6 == 80 || $6 == 443 {print}' k.h

 **Filter out rows where Destination\_Port contains alphanumeric characters:**

awk -F"," '$6 !~ /[a-zA-Z]/ {print}' k.h

 **Filter out traffic where Protocol is TCP and Destination\_Port is 443:**

awk -F"," '!($4 == "TCP" && $6 == 443) {print}' k.h

 **Filter out and print traffic where Packet\_Size > 1000 OR Status is Dropped:**

awk -F"," '$7 > 1000 || $8 == "Dropped" {print}' k.h

 **Print traffic NOT originating from 192.168.x.x IP addresses:**

awk -F"," '$2 !~ /^192\.168\./ {print}' k.h

 **Filter rows where both Source\_IP and Destination\_IP are in the 192.168.x.x range:**

awk -F"," '$2 ~ /^192\.168\./ && $3 ~ /^192\.168\./ {print}' k.h

 **Filter out traffic where Destination\_Port is 22 OR Packet\_Size < 100 bytes:**

awk -F"," '!($6 == 22 || $7 < 100) {print}' k.h

**1. write a shell script to get value the pattern and file name from the user and check the pattern exists or not. If the pattern exists print the relavent message , if pattern not found print relevent message.**

#!/urs/bin/bash

read -p "Enter pattern : " pattern

read -p "Enter file name : " file

if grep -q "$pattern" "$file"; then

echo "The pattern '$pattern' was found in the file '$file'."

else

echo "The pattern '$pattern' was not found in the file '$file'."

fi

**2. Modify the above script to pass the arguments from command line arguments.**

#!/urs/bin/bash

pattern = "$1"

file = "$2"

if grep -q "$pattern" "$file"; then

echo "The pattern '$pattern' was found in the file '$file'."

else

echo "The pattern '$pattern' was not found in the file '$file'."

fi

**3. Modify the above script to pass the values inside the script.**

#!/urs/bin/bash

pattern = "abc"

file = "text.txt"

if grep -q "$pattern" "$file"; then

echo "The pattern '$pattern' was found in the file '$file'."

else

echo "The pattern '$pattern' was not found in the file '$file'."

fi

**4. validate the script (script 1, script 2)**

**- the file exists or not**

#!/urs/bin/bash

read -p "Enter pattern : " pattern

read -p "Enter file name : " file

if [ ! -f "$file" ]; then

echo "The file '$file' does not exist."

exit 1

if grep -q "$pattern" "$file"; then

echo "The pattern '$pattern' was found in the file '$file'."

else

echo "The pattern '$pattern' was not found in the file '$file'."

fi

**- arguments passed or not**

#!/urs/bin/bash

if [ "$#" -ne 2 ]; then

echo "missing paramater"

exit 1

fi

pattern = "$1"

file = "$2"

if grep -q "$pattern" "$file"; then

echo "The pattern '$pattern' was found in the file '$file'."

else

echo "The pattern '$pattern' was not found in the file '$file'.

fi