-A Cutting Columns.
To extract specific Column you need to follow the '-c' option with a list Of Column numbers, ~~embe~~ Delimited (Seperated) by a Comma. Ranges can also be used with hyphen (-).

```
cut -c 6-24, 28-32
```

The '-c' option is useful for fixed length Lines. Most unix files do not contain fixed-ed length Lines.

To extract useful Data from such files you need to cut fields rather than columns.

Cut uses 'tab' as default field Delimiter
Two options need to be used there.
'-d' for field delimiter
'-f' for field list.

---

paste: What you have cut with cut Command can be pasted back with paste Command but vertically rather than horizontally. you can view 2 files Side by side by pasting them.

```
$ Cat cut1
$ Cat cut2
$ paste cut1 cut2.
```

Sort : Ordering a file.
Sorting is the ordering of Data in ascending or decending Sequence by Default 'Sort' reorders lines in ASCII-Colatting sequence . white space first, then ~~nt~~ numerals, upper case letters and finally lower case letters.
To sort on the specified field number.
'-r' to short in reverse order.

Uniq : It is the command line utility that reports or filter outs the reported line in a file. It is a tool that helps to Defect the adjacent duplicate lines and also deletes the duplicate line.

Scanned by CamScanner

1/3/19 (Friday)

in sort command
white space
~~[illegible]~~ number
capital
small.

uniq command :

eg
```
$ cat uni.txt
  I love music.
  I love music.
  "

  I love music of kartik.
  I love music of kartik.

  ~~Tarnt~~
  Thanks.
$ uniq uni.txt
  I love music.
  I love music of kartik.
  Thanks.
```

option (1) (-d) is used to print duplicate line.

```
$ uniq -d uni.txt
```

(2) (-u) allows to print only unique line.

```
$ uniq -u uni.txt
```

(3) (-i) is used to ignore case.

```
$ uniq -i uni.txt
```

```
$ sort -u uni.txt
  I love music
  I love music of kartik.
  Thanks.
```

→ when (-u) option is used with sort command it will work like ~~wit~~ unique command but it will display the repeated lines uniquely in sorted order.

```
$ uniq uni.txt
  I love music

  I love music of kartik.

  Thanks.
```

• __tr__ : __command (translating character)__

-8 The tr filter manipulates individual charact in a line.

More specificly, it translates character using one or two compact expression.

```
$ tr option Expression1 Expression2 Standard-Inpu
```

eg `$ wc emp.txt` ⇒ This is not standard Input

`$ wc < emp.txt` ⇒ standard input

Expression1 → Character or set of characters needed to changed

Expression2 → character with whome it will be translated

```
eg. $ tr 'a' 'A' < emp.txt
```

eg. `tr '|' '@' < shortlist`

```
 . . . @ . — — @
 . . . @ . — — @
 . . . @ . — . @
```

eg. `$ head -n3 shortlist | tr '[a-z]' '[A-Z]'`

__~~Note~~__:

```
. . . | A.K. | . . .
      | B.K. | . . .
```

__Note__: tr takes input only from standard input, it does not take file name as argument.

By default it translates each character in expression 1 on to its maped counter part in expression 2.

Q. write a command

1 – 20 → 5, 10 – only changed its letter.

(-d) opion :

eg. $ tr -d '1' <shortlist

To delete a particular character '-d' option is used.

1. grep command : grep (stands for globally search for regular expression and print) is a powerful tool that searches a file for a particular pattern of character and displays all the lines that contain that pattern. The pattern that is search in file is called regular expression.

Syntax $\boxed{\text{\$ grep [option] pattern [File or No. of File]}}$

```
$ cat myfile.txt
Unix Is a OS .. Unix is easy to learn.
```

$ grep "Unix" myfile.txt → $\boxed{\text{it is case sensitive}}$

```
Unix is a OS. Unix is easy to learn.
```

(i) (-i) option : is used to ignore case.

~~$ grep -i" Unix"~~

```
$ grep -i "Unix" myfile.txt
```

* ~~def~~ regular expression is desired pattern defined by user.

Q. Use ls and grep command to show only directory.

(ii) (-c) option : print only a count of the lines that match a pattern.

```
$ grep -c "director" emp.txt
4
```

(iii) (-h) option: displays the match lines but donot display the file name.

(iv) (-l) option: display list of file names only.

eg. let Abc is director has too many files and we want to show word "Unix" in all file so we used (-l) option and we should also used (-i) option.

(V) (-n) option : displays the matched lines and their line no.

m.I. (vi) (-v) option: This print out all the line that do not match the ~~mat~~ pattern.

(vii) (-w) option: Matches hole word.

(viii) (-o) option: Prints only matched part of matching line, with each such part on a sepperate output.

Scanned by CamScanner

6/3/19 (Wednesday)

Regular Expression:- Regular expression provides an ability to match a "string of text" in a very flexible and concise manner.
A string of text can be further defined as a single character, word, sentence or particular pattern.

```
$ grep [option] [pattern] file(s)
```

Option (-RE)

$$\begin{tikzcd} & \text{(-RE)} \arrow[dl] \arrow[dr] & \\ \text{Basic} & & \text{Extended} \end{tikzcd}$$

```
$ grep -E
```

Regular expression take cares of some common queries and substitution requirement.

Regular expression belongs to 2 categories :-

(i) Basic
(ii) Extended

grep support basic regular expression by default and extended regular expression with **-E** option.

or **$ egrep**

mi

(i) Basic Regular Expression :-

(1) The Character Class :- A regular expression lets you specify a group of character in close within a pair of [ ] bracket.

eg. A single character p, q or r. ⇐ [pqr]

[c_1-c_2] ⇒ A single character within the ASSCI range represented by . [a-f]

[1-5] ⇒ Any digit from 1 to 5.

[^abc] ⇒ A single character which is not a, b, c .

eg. $ grep "[Aa]" emp.txt
$ grep "[Aa]g[ar]" emp.txt
$ grep "[aA]g[ar] [ar]wal" emp.txt

eg. $\overset{a}{A}$ g $\overset{a}{r}$ $\overset{a}{r}$ wal

(2) The * :- It specifies $\overset{\text{'zero'}}{0}$ or more occurrences of the previous character.

* ⟶ zero or more

eg. $ grep "[Aa]gg*[ar]wal" emp.txt

(3) The dot (•) :- A dot (•) snaches a single characters.

eg. $ grep "j.*saxena" emp.txt
eg $ grep "U.*" file name
eg $ grep "U." file name
eg. $ grep "U*" file name

(4) Specifying Pattern location:- Two symboles are use for matching the pattern location.

eg. ^(caret) ⇒ For matching at the begining of the line

$(dollar) ⇒ For matching at the end of the line.

eg ^cap

cap$

Q. To print the line where unix is started in first.

eg ~~$ grep~~ $ grep "^unix" filename.

Q. List all the files which ends with sh.

Ans $ ls -l | grep "sh$"

Q. List all the files which name start with a.

read × $ ls -l | grep " ^ a "

Q. print line which start with 2.

$ grep "^2" emp.txt

7/3/19 Thursday

$grep "7... $" emp.txt   show employee salary start with 7000.

Show only directories:-

$ ls -l | "^d"

Scanned by CamScanner

[ii] Extended Regular Expression and ~~E~~ egrep

Extended Regular Expression makes ~~impossible~~ in possible to match dissimilar pattern with a single expression. This sets uses some additional expression and POSIX compliant versions of grep use them with -E option. If your version of grep does not support this option then used egrep. But without ~~-d~~ -E option.

$\boxed{\text{\$ grep -V}}$ this command shows version.

1. The + and ? :-

+ ⇒ Matches one or occurences of the previous character.

? ⇒ Matches 0 or 1 occurences of previous character.

```
$ grep -E "[aA]gg?arwal" emp.txt
$ egrep "[aA]gg?arwal" emp.txt
```
both work same

2. Matching Multiple Pattern ~~&~~ (|, (and)) :-

The | (pipe) is delimiter of multiple pattern.

eg. exp1 | exp 2 ⇒ it matches exp1 or exp2

```
eg. $ egrep "sengupta | dasgupta" emp.txt
eg  $ egrep "(sen|das)gupta" emp.txt
```
both work same

```
$ ls -l|grep"a[0-9]x"
```

Sed command (~~string~~ stream editor)

- This command do multiple task.
- Sed is a multipurpose tools which combined the work of several filter.
- It is derived from ed (editor), original UNIX editor.
- Sed perform non-interactive operations on a data strings.
- sed command is in UNIX is stands for stream editorand it can perform lots of functions like searching, find and replace, insertion or deletion.
- The most comman use of sed command is for substitution or for find and replace.
- By using sed you can edit files even without openning it.
- It is more faster than other any option.

→ Sed uses instruction to act on text.

→ An instruction combined an address for selecting line, with an action to be taken on them.

Syntax

$\boxed{\text{\$ sed 'address action' file(s)}}$

The address and action are inclosed within single qotes ' '.

Addressing in sed is done in 2 ways.

1. By one or two line numbers:- ~~[illegible]~~ (line 3,7)
2. By specifing / enclosed pattern which occurress in a line.

eg.