Cut : used to extract sections **from each line of input**

```
cut [-b] [-c] [-f list] [-n] [-d delim] [-s] <filename>
```

Arguments	Descriptions	Examples
-b	select only these bytes	
-c	select only these characters	-c1 -c1-5 -c3- -c-8 (single value of range)
-f	select only these fields (columns)	-f1,2 -f 3 - f1-2,4-5
-d	specify delimiter (tab by default)	-d":" -d" " -d","
-s	do not print lines not containing delimiters	

Head : used to read the **first** few lines or characters from input

```
head [options] <filename>
```

Arguments	Descriptions	Examples
	default: print the first 10 lines	
-n	specify the number of lines	-n2 -n 30
-c	specify the number of bytes	-c1 -20 (print the first c chars from input)

Tail : used to read the **last** few lines or characters from input

```
tail [options] <filename>
```

Arguments	Descriptions	Examples
	default: print the last 10 lines	
-n	specify the number of lines	-n2 -n 30
-c	specify the number of bytes	-c1 -20 (print the first c chars from input)
-f	follow the last changes made in file	tail -f file.txt

Tr : used to "translate" (replacing or removing specific characters)

```
tr [options] SET1 [SET2]  <filename>
```

Arguments	Descriptions
-d	delete
-s	consider all consecutive occurrences
-c	use the complement of SET1

Examples	Descriptions
tr 'H' 'J'	replace 'H' characters with 'J'
tr [:lower:] [:upper:], tr [a-z] [A-Z]	convert lower case to upper case
tr **-d** [0-9] , tr **-d** [:digit:]	delete all digits (and therefore all numbers)
tr [:space:] '\t'	replace white spaces with tabs
tr **-s** [:space:] '\t'	replace multiple white spaces with one tab
tr **-c -d** [:digit:]	delete anything **BUT** the digits
tr **-cd** [:print:]	remove all **non-printable** characters
tr -s '\n' ' '	replace new lines with spaces (join all lines into a single line)

Sort : sort the lines or the specified fields of input

```
sort [options] <filename>
```

Arguments	Descriptions
-K	consider only these fields
-t	specify delimiter (tab by default)
-r	reverse the sorting order
-n	sort by numbers
-u	unique: remove duplicate records
-f	ignore case
-b	ignore leading blanks
-i	consider only printable characters
-c	check to see if input is sorted
-o <file>	redirect the output to a file

Examples	Descriptions
sort -t'|' +2-4	sort by the 3rd and the 4th columns
sort -k 3.3,3.5	sort based on only the third-through-fifth characters of the third field of each line

Uniq : reports or filters out repeated lines in a file

```
uniq [options] <filename>
```

Arguments	Descriptions
-c	count number of occurrences
-d	print only lines that have multiple occurrences (duplicates)
-D	print lines that have duplicates (including the duplicate lines)
-u	print lines with only one occurrence (not repeated)
-w	compare only the first *w* characters
-s	skip comparing the first s characters
-f	skip comparing the first s fields (the default delimiter is the space)

Examples	Descriptions
uniq -c -w 8	print the number of lines that have the same first 8 characters
uniq -d -s 2	print the duplicate lines: comparison made on the characters from position '3'

Paste : join files horizontally (parallel merging)

```
paste [options] <filename1> [<filename2>]
```

Arguments	Descriptions
-d	delimiter (by default tab)
-s	append the data in serial rather than in parallel

Examples	Descriptions
paste -d, -s file1	join all lines using the comma delimiter
paste - - < file1	merge a file by pasting the data into 2 columns
paste -d ':,' - - - < file1	merge a file into 3 columns using 2 different delimiters
paste file1 file2	paste contents of 2 files side by side
paste -d'\n' file1 file2	read lines in both the files alternatively

Sed : Stream Editor used for modifying text & files

```
sed [options]... [SCRIPT] [<filename>]

General syntax: action/pattern1/pattern2/
```

Actions	Descriptions
s	substitute (replace)
d	delete
p	print

Examples	Descriptions
sed '**s**/pattern1/pattern2/'	substitute only the **first** occurrence of pattern1 with pattern2
sed '**s**/pattern1/pattern2/**2**'	substitute only the **second** occurrence of pattern1 with pattern2
sed '**s**/pattern1/pattern2/**g**'	substitute **all** occurrences of pattern1 with pattern2 (g=global)
sed '**s**/pattern1/pattern2/3**g**'	substitute pattern1 with pattern2 starting **from the 3rd** occurrence
sed '**s**/http:\/\//www/'	substitute "http://" with "www"
sed '**s**/the/THE/**ig**'	substitute all occurrences of "the" with "THE", **case insensitive** (i)
sed '**2 s**/the/THE/'	replace only on the **second line**
sed '**3,$ s**/the/THE/'	replace only from the **third line until EOF**
sed '**3,100 s**/the/THE/'	replace only from the **3rd to the 100th line**
sed **-e '...' -e '...'**	**apply two commands**
sed '/hi/ **s**/the/THE/g'	replace "the" with "THE" only on lines matching pattern "hi"
sed 's/\([0-9]*\) meters/**\1** m/gi'	\1 = retrieve the value of the match ("()")
sed 's/**\b**the**\b**/THE/g'	substitute the **word** "the" with "THE" (\b = word boundary)
sed 's/[0-9]/&&/g'	repeat digits

Grep : global search using regular expression

```
grep [options] pattern [<filename>]
```

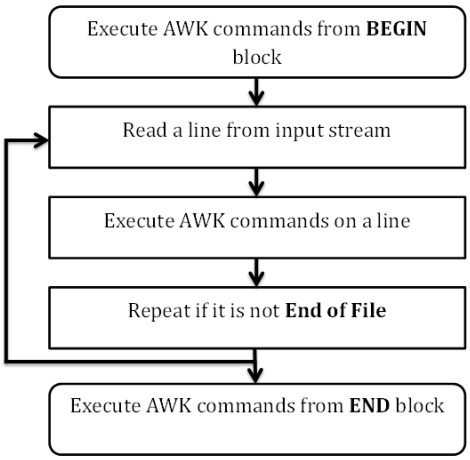
Options	Descriptions
-n	show the line numbers of matching lines
-i	ignore **case** in matching patterns
-w	select lines that match whole **words**
-v	invert the sense of matching, to select non-matching lines
-o	show only the matched string (instead of the entire line)
-c	suppress output, just **count** the number of lines matching the pattern
--color	successful matches will be highlighted
-r	search recursively in all subdirectories

Awk : interpreted programming language specially designed for the text processing

```
awk pattern {action} [<filename>]
```

```
BEGIN {awk-commands}
```

The begin block (*optional*) gets executed at program startup and is executes only once. This is good place to initialize variables. BEGIN is the AWK keyword and hence it must be in upper case.




```
/pattern/ {awk-commands}
```

The body block apply AWK commands on every input line. By default AWK execute commands on every line but we can restrict this by providing pattern.

```
END {awk-commands}
```

The end block (*optional*) gets executed at the end of program. END is the AWK keyword and hence it must be in upper case

Options	Descriptions	Example
-v	assign a value to a variable	awk **-v** name=Jerry 'BEGIN{printf "Name = %s\n", name}'
-F	delimiter	awk -F":" '{ print $1 $3 }' marks.txt

Examples:

- display the file contents with header by using AWK script.

```
awk 'BEGIN{printf "Sr No\tName\tSub\tMarks\n"} {print}' marks.txt
```

- print column or field

```
awk '{print $3 "\t" $4}' marks.txt
```

- print all lines that match pattern

```
awk '/a/ {print $0}' marks.txt    OR    awk '/a/' marks.txt    // $0 = entire input
```

- print lines having more than 18 characters

```
awk '/a/{++cnt} END {print "Count = ", cnt}' marks.txt
```

- print lines having more than 18 characters

```
awk 'length($0) > 18' marks.txt
```

- ignore case

```
awk 'BEGIN{IGNORECASE=1} /amit/' marks.txt
```

- math

```
awk 'BEGIN { a = 50; b = 20; print "(a + b) = ", (a + b) }'
awk 'BEGIN { a = 50; b = 20; print "(a % b) = ", (a % b) }'
awk 'BEGIN { param = 5.5; result = log (param); printf "log(%f) = %f\n", param, result}'
```

- comparisons

```
awk 'BEGIN { a = 10; b = 10; if (a == b) print "a == b" }'
awk 'BEGIN { a = 10; b = 20; if (a != b) print "a != b" }'
awk 'BEGIN { a = 10; b = 20; if (b > a ) print "b > a" }'
awk 'BEGIN {num = 5; if (num >= 0 && num <= 7) printf "%d is in octal format\n", num }'
awk 'BEGIN {ch = "\n"; if (ch == " " || ch == "\t" || ch == "\n") print "Current character is whitespace." }'
awk 'BEGIN { name = ""; if (! length(name)) print "name is empty string." }'
awk 'BEGIN { a = 10; b = 20; (a > b) ? max = a : max = b; print "Max =", max}'
```

- string

```
awk 'BEGIN { str1="Hello, "; str2="World"; str3 = str1 str2; print str3 }'
awk 'BEGIN { str = "Hello, World !!!"; print "Length = ", length(str)}'
awk 'BEGIN { str = "One Two"; subs = "Two"; ret = index(str, subs); printf "Sub \"%s\" found at poz=%d\n", subs, ret}'
awk 'BEGIN { str = "One Two"; subs = "Two"; ret = match(str, subs); printf "Sub \"%s\" found at poz=%d.\n", subs, ret}'
awk 'BEGIN { str = "One,Two"; split(str, arr, ","); print "Array: "; for (i in arr) { print arr[i] } }'
awk 'BEGIN { str = "Hello, World"; sub("World", "Jerry", str); print "String after replacement = " str}' // one replace
awk 'BEGIN { str = "Hello, World !!!"; subs = substr(str, 1, 5); print "Substring = " subs}'
awk 'BEGIN { str = "HELLO, WORLD !!!"; print "Lowercase string = " tolower(str)}'
awk 'BEGIN { str = "HELLO, WORLD !!!"; print "Lowercase string = " toupper(str)}'
```

- array

```
awk 'BEGIN { arr[0] = 1; arr[1] = 2; arr[2] = 3; for (i in arr) printf "arr[%d] = %d\n", i, arr[i] }'
awk 'BEGIN {fruits["mango"]="yellow"; fruits["orange"]="orange" print fruits["orange"] "\n" fruits["mango"]}'
awk 'BEGIN {fruits["mango"]="yellow"; fruits["orange"]="orange"; delete fruits["orange"]; }'
```

- regular expressions

```
awk '$0 ~ 9' marks.txt       // lines which contain the character '9'
awk '$0 !~ 9' marks.txt      // lines which do not contain the character '9'
awk '/f.n/' marks.txt        // lines that match strings like "fun", "fan", ... (. = one character)
awk '/[CT]all/' marks.txt    // lines that match strings like "Call", "Tall", ...
awk '/^The/'                 // lines that start with "The"
awk '/n$/'                   // lines that finish with "n"
awk '/Call|Ball/'            // lines that contain "Call" or "Ball"
awk '/Colou?r/'              // "?" = 0 or 1 occurrence
awk '/cat*/'                 // "*" = 0 or more occurrence
awk '/2+/'                   // "+" = 1 or more occurrence
awk '/Apple (Juice|Cake)/'  // grouping => "Apple Juice, Apple Cake
```

- Loop

```
awk 'BEGIN { for (i = 1; i <= 5; ++i) print i }'
awk 'BEGIN {i = 1; while (i < 6) { print i; ++i } }'
awk 'BEGIN {sum = 0; for (i = 0; i < 20; ++i) { sum += i; if (sum > 50) break; else print "Sum =", sum } }'
awk 'BEGIN {for (i = 1; i <= 20; ++i) {if (i % 2 == 0) print i ; else continue} }'
```