**Kill Command in Unix/Linux**

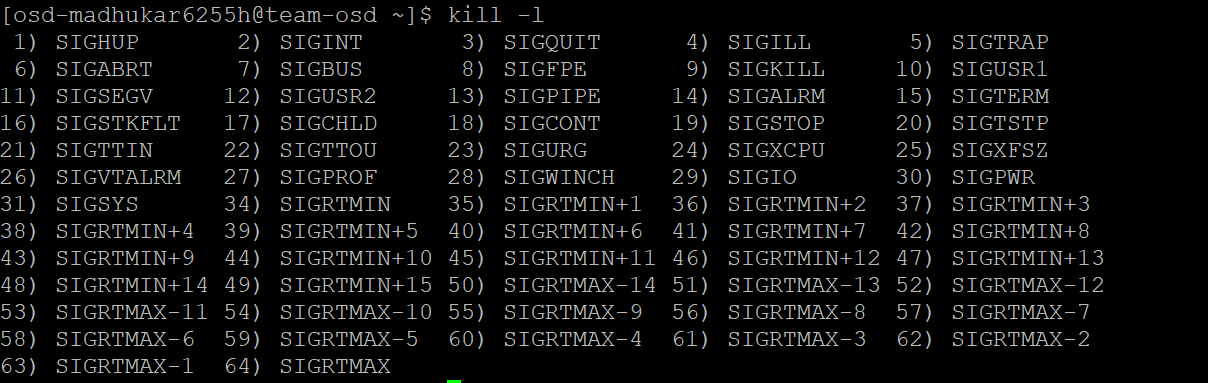
kill command in Linux (located in /bin/kill), is a built-in command which is used to terminate processes manually. kill command sends a signal to a process which terminates the process. If the user doesn’t specify any signal which is to be sent along with kill command then default TERM signal is sent that terminates the process.

**Options and examples**

1. **kill -l :**To display all the available signals you can use below command option:

**Syntax:**

$kill -l



* **By number (e.g. -5)**
* **With SIG prefix (e.g. -SIGkill)**
* **Without SIG prefix (e.g. -kill)**

**Note:**

* Negative PID values are used to indicate the process group ID. If you pass a process group ID then all the process within that group will receive the signal.
* A PID of -1 is very special as it indicates all the processes except kill and init, which is the parent process of all processes on the system.
* To display a list of running processes use the command *ps* and this will show you running processes with their PID number. To specify which process should receive the kill signal we need to provide the PID.

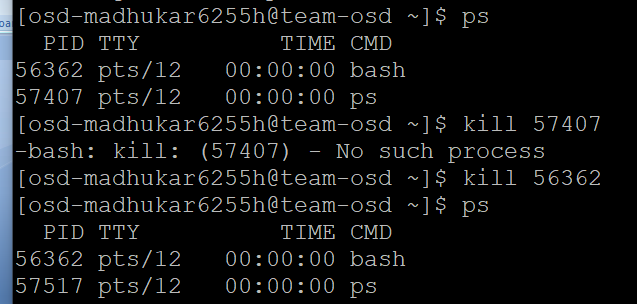
**Syntax:**

$ps

**2. kill pid :** To show how to use a PID with the kill command.

**Syntax:**

$kill pid



**3. kill -s :** To show how to send signal to processes.

**Syntax:**

kill {-signal | -s signal} pid

**4. kill -L :**This command is used to list available signals in a table format.

**Syntax:**

kill {-l | --list[=signal] | -L | --table}

# grep command in Unix/Linux

The grep filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern. The pattern that is searched in the file is referred to as the regular expression (grep stands for globally search for regular expression and print out).

**Syntax:**

**grep [options] pattern [files]**

**Options Description**

**-c** : This prints only a count of the lines that match a pattern

**-h :** Display the matched lines, but do not display the filenames.

**-i :** Ignores, case for matching

**-l :** Displays list of a filenames only.

**-n :** Display the matched lines and their line numbers.

**-v :** This prints out all the lines that do not matches the pattern

**-e exp :** Specifies expression with this option. Can use multiple times.

**-f file :** Takes patterns from file, one per line.

**-E :** Treats pattern as an extended regular expression (ERE)

**-w :** Match whole word

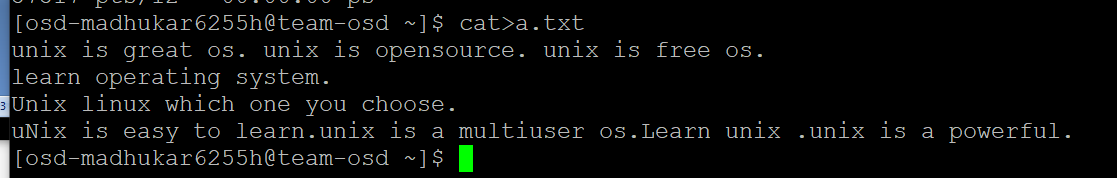
**-o :** Print only the matched parts of a matching line,

with each such part on a separate output line.

**Sample Commands**

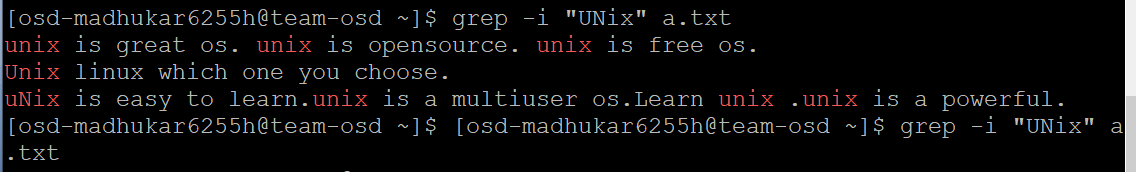
Consider the below file as an input.

**$cat > a.txt**



**1. Case insensitive search:**The -i option enables to search for a string case insensitively in the given file. It matches the words like “UNIX”, “Unix”, “unix”.

**$grep -i "UNix" a.txt**

**Output:** 

**2. Displaying the count of number of matches :** We can find the number of lines that matches the given string/pattern

**$grep -c "unix" a.txt**

**Output:**

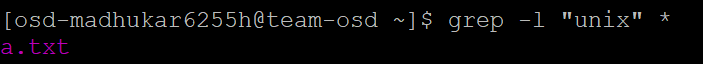
 **3. Display the file names that matches the pattern:**We can just display the files that contains the given string/pattern.

**$grep -l "unix" \***

**or**

**$grep -l "unix" f1.txt f2.txt f3.xt f4.txt**

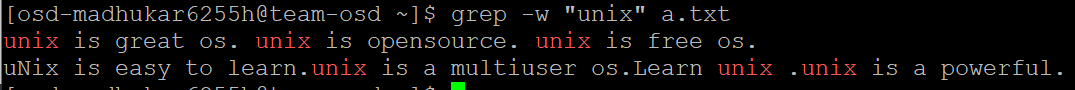
**Output:**



**4. Checking for the whole words in a file:**By default, grep matches the given string/pattern even if it found as a substring in a file. The -w option to grep makes it match only the whole words.

**$ grep -w "unix" a.txt**

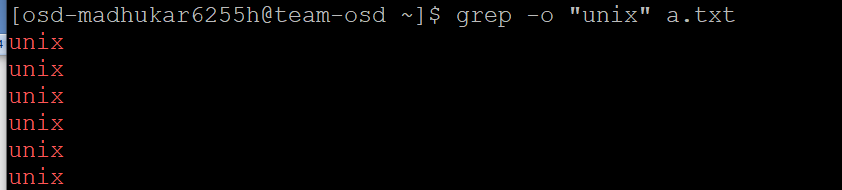
**Output:**



**5. Displaying only the matched pattern:**By default, grep displays the entire line which has the matched string. We can make the grep to display only the matched string by using the -o option.

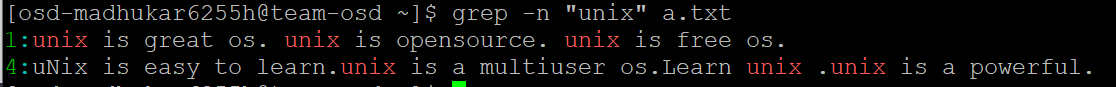
**$ grep -o "unix" a.txt**

**Output:**

 **6. Show line number while displaying the output using grep -n :**To show the line number of file with the line matched.

**$ grep -n "unix" a.txt**

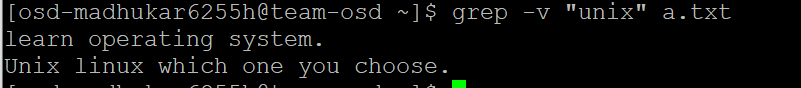
**Output:**



**7. Inverting the pattern match :**You can display the lines that are not matched with the specified search sting pattern using the -v option.

**$ grep -v "unix" a.txt**

**Output:**

**8. Matching the lines that start with a string:**The ^ regular expression pattern specifies the start of a line. This can be used in grep to match the lines which start with the given string or pattern.

**$ grep "^unix" a.txt**

**Output:**

**9. Matching the lines that end with a string:**The $ regular expression pattern specifies the end of a line. This can be used in grep to match the lines which end with the given string or pattern.

**$ grep "os$" a.txt**

**grep.c**

// Simple grep. Only supports ^ . \* $ operators.

#include "types.h"

#include "stat.h"

#include "user.h"

char buf[1024];

int match(char\*, char\*);

void

grep(char \*pattern, int fd)

{

int n, m;

char \*p, \*q;

m = 0;

while((n = read(fd, buf+m, sizeof(buf)-m)) > 0){

m += n;

p = buf;

while((q = strchr(p, '\n')) != 0){

\*q = 0;

if(match(pattern, p)){

\*q = '\n';

write(1, p, q+1 - p);

}

p = q+1;

}

if(p == buf)

m = 0;

if(m > 0){

m -= p - buf;

memmove(buf, p, m);

}

}

}

int

main(int argc, char \*argv[])

{

int fd, i;

char \*pattern;

if(argc <= 1){

printf(2, "usage: grep pattern [file ...]\n");

exit();

}

pattern = argv[1];

if(argc <= 2){

grep(pattern, 0);

exit();

}

for(i = 2; i < argc; i++){

if((fd = open(argv[i], 0)) < 0){

printf(1, "grep: cannot open %s\n", argv[i]);

exit();

}

grep(pattern, fd);

close(fd);

}

exit();

}

// Regexp matcher from Kernighan & Pike,

// The Practice of Programming, Chapter 9.

int matchhere(char\*, char\*);

int matchstar(int, char\*, char\*);

int

match(char \*re, char \*text)

{

if(re[0] == '^')

return matchhere(re+1, text);

do{ // must look at empty string

if(matchhere(re, text))

return 1;

}while(\*text++ != '\0');

return 0;

}

// matchhere: search for re at beginning of text

int matchhere(char \*re, char \*text)

{

if(re[0] == '\0')

return 1;

if(re[1] == '\*')

return matchstar(re[0], re+2, text);

if(re[0] == '$' && re[1] == '\0')

return \*text == '\0';

if(\*text!='\0' && (re[0]=='.' || re[0]==\*text))

return matchhere(re+1, text+1);

return 0;

}

// matchstar: search for c\*re at beginning of text

int matchstar(int c, char \*re, char \*text)

{

do{ // a \* matches zero or more instances

if(matchhere(re, text))

return 1;

}while(\*text!='\0' && (\*text++==c || c=='.'));

return 0;

}