Table of Contents

find

Standard Unix utility which searches for files
The find utility selects files that are located in specified directory hierarchies and that meet specified criteria.
syntax:

find dirList criteria action
find [directory-list] [option] [expression]

- The directory list specifies the directory hierarchies that find is to search. When you don't specify this, find searches the working directory hierarchy.
- The option/criteria controls whether find dereferences symbolic links as it descends directory hierarchies. By default, find doesn't dereference symbolic links.
- The expression/action contains criteria for filtering files. When you don't specify an expression, it defaults to -print.

Basically
find [starting directory] [additional conditions for the search] [criteria for filtering/locating files]

The criteria can use the following tests:
-name *namePattern* file names matching the namePattern
-iname *namePattern* file names matching the namePattern, ignoring case
-user *name* file owner matches this user name or name can be a numeric user ID

A space separating two criteria is a boolean AND operator. The file has to meet both criteria to be selected. A -or || -o separating the criteria is a Boolean OR operator.
You can negate any criterion by preceding it with an exclamation point. The find utility evaluates criteria from left to right unless you group them using parenthesis.
With the expression, you need to quote special characters so the shell doesn't interpret them, only passes them to find. Special characters frequently used include: '(', ')', '!', criterion, or other element.

Symbolic Link/Symlink/Soft link
This is a type of file that acts as a pointer to another file or directory in the file system. It works similarly to a shortcut on windows.

Linux Manual:
A symbolic link is a special type of file whose contents are a string that is the pathname of another file, the file to which the link refers. (The contents of a symbolic link can be read using readlink(2)) In other words, a symbolic link is a pointer to another name, and not to an underlying object. For this reason, symbolic links may refer to directories and may cross filesystem boundaries.

Options (Controls the treatment of symlinks)

-H Don't follow symlinks, except while processing the command line arguments. When **find** examines or prints information about files, the information used shall be taken from the properties of the symbolic link itself.

-L Follow symlinks. When **find** examines or prints information about files, the information used shall be taken from the properties of the file to which the link points, not from the link itself

-P (default) Never follow symlinks. When find examines or prints information about files, and the file is a symbolic link, the information used shall be taken from the properties of the symbolic link itself.

(-P or default)
find /path -name "*.txt"

With Dereferencing (-L)
find -L /path -name "*.txt"

☆ If more than one of the options is specified, each overrides the others; the last one appearing on the command line takes effect.

Criteria

☆ ±n is a decimal int that can be expressed as +n (more than n), -n (fewer than n), or n (exactly n).

-anewer filename
(accessed newer) the file being evaluated; if it was accessed more recently than filename.

-atime ±n
(access time) if it was last accessed ±n days ago. when you use this option, find changes the access times of directories it searches.

-depth
the file being evaluated always meets this criterion. it causes find to take action on entries in a directory before it acts on the directory itself.

`-exec command\;`
if the command returns a 0 exit status. you have to terminate the command with a quoted semicolon. The find utility replaces a pair of braces ({}) within the command with the name of the file being evaluated. you can use the -exec action at the end of a group of other criteria to execute the command if the preceding criteria are met.

`-group name`
if it's associated with the group named 'name'. you can use a numeric group ID instead.

`-inum n`
the file meets this criterion if its inode number is n.

`-links ±n`
the file meets this criterion if it has ±n links

`-mtime ±n`
(modify time) meets criteria if it was last modified ±n days ago.

`-name filename`
The pattern 'filename' matches its name. it can include wildcard characters ('*', '?', and '[]') but these characters need to be quoted.

`-newer filename`
The file was modified more recently than 'filename'

`-nogroup`
The file doesn't belong to a group known on the local system.

`-nouser`
The file doesn't belong to a user known on the local system.

`-ok command\;`
Same as -exec except it displays each command to be executed enclosed in angle brackets as a prompt and executes the command only if it receives a response that starts with a y/Y from standard input

`-perm[±]mode`
The file has access permissions given by mode. If mode is preceded by '-' the file access permissions must include all the bits in mode. (e.g if mode is 644 a file with 755 permissions will meet this criterion). If mode has '+', the file access permissions must include at least one of the bits in mode. If there's no +/- then the mode of the file must exactly match mode. You can use either symbolic or octal rep for mode.

`-print`
The file always meets this criterion and when it does, find displays the pathname of the file it's evaluating. If -print is the only criterion in the expression, find displays the names of all files in the directory-list. If it appears with other criteria, find displays the name only if the preceding criteria are met. If there's no action criteria, -print is assumed.

`-size ±n[clk|M|G]`
if the file is the size specified by ±n, measured in 512-byte blocks. Follow 'n' with the letter 'c' to measure files in characters, 'k' to measure files in kilobytes, 'M' to measure in megabytes, or 'G' to measure in gigabytes.

`-type filetype`
if the file type is specified by:
`b` - block special file
`c` - character special file
`d` - directory file
`f` - ordinary file
`l` - symbolic link
`p` - FIFO (named pipe)
`s` - socket

`-user name`
if the file belongs to the user with the username 'name'. You can use a numeric user ID in place of 'name'.

`-xdev`
always meets this criterion - Prevents find from searching directories in file systems other than the one specified by 'directory-list'. Also -mount.

`-x`
always meets this criterion - prevents find from searching directories in the file systems other than the one specified by 'directory-list'. Also -mount.

Example:
[Assuming x and y are criteria]

```
$ find dir x y
```

The command line never tests whether the file meets criterion y if it doesn't meet criterion x due to the criteria being separated by a space (&&).

```
$ find dir x -or y
```

"Test whether the file meets criterion x or criterion y."

```
$ find dir -print -links +1
#displays the names of all files in the directory and its
subdirectories regardless of how many links they have:
$ find /home/res205 -print -links +1
/home/res205
/home/res205.vimrc
/home/res205/courses
/home/res205/courses/cs3423
#and so on..

#displays the names of only those files in the directory that have more
than one link:
$ find /home/res205 -links +1 -print
/home/res205
/home/res205/courses
/home/res205/courses/cs3423
/home/res205/courses/cs3423/find
/home/res205/courses/cs3423/find/Pgm1
/home/res205/courses/cs3423/find/Pgm5
/home/res205/courses/cs3423/find/Pgm2
/home/res205/courses/cs3423/myconfig
/home/res205/courses/cs3423/myconfig/vim
/home/res205/courses/cs3423/Spring25
/home/res205/courses/cs3423/Spring25/assign2
/home/res205/courses/cs3423/Spring25/assign1
/home/res205/courses/cs3423/IPC
/home/res205/courses/cs3423/grep
/home/res205/courses/cs3423/grep/Program4
/home/res205/courses/cs3423/grep/Program5

$ find /home/res205 -links +1
#generates the same output as above, this is to display how -print is
the default whether or not it's specified.

$ find
#The simplest find command has no arguments and lists the files in the
working directory & all subdirectories.
```

```
$ find . -name ‘a*’ -print
#finds and displays the pathnames of files in the working directory and
subdirectories that have filnames beginning with ‘a’.
#’.’ designates the working directory
./courses/cs3423/myconfig/vim/append_to_vimrc.txt
./courses/cs3423/Spring25/assign2
./courses/cs3423/Spring25/assign2/assign2.sed
./courses/cs3423/Spring25/assign2/assign2.bash
./courses/cs3423/Spring25/assign2/a2-resetdata.bash

$ find -name ‘a*’
#same output as above

$ find / -type f -mtime -1 ! -name ‘*.o’ -print
#“find, in the root directory and all subdirectories (/), ordinary files
#(-type f) that have been modified within the past day (-mtime -1), with the
#exception of files whose names are suffixed with .o (! -name ' * .o').”
/proc/209/cmdline
/proc/209/stat
etc. . .

$ find . \( -name core -o -name junk\) -print -exec rm {} \;
#finds, displays the filenames of, and deletes the files named core or junk in
#the working directory & its subdirectories.

$ find /home/max /home/hls -size +100 -atime +5 -ok rm {} \;
< rm . . . /home/max/notes >? y
< rm . . . /home/max/letter >? n
#looks in 2 user directories for files that are larger than 100 blocks (-size
+100),files that haven’t been accessed within the past five days (-atime +5).
Then the find command asks if you want to delete the file (-ok rm {}).

$ ls -l /home/sam/track
lrwxrwxrwx. 1 sam pubs  15 04-12 10:35 memos -> /home/sam/memos
-rw-r--r--. 1 sam pubs 12753 04-12 10:34 report
#/home/sam/track is a symbolic link to the directory named /home/sam/memos
#using -L, find dereferences (follows) the symbolic link and searches the dir.

$ find /home/sam/track
/home/sam/track
/home/sam/track/memos
/home/sam/track/report

$ find -L /home/sam/track
/home/sam/track
/home/sam/track/memos
/home/sam/track/memos/memo.710
/home/sam/track/memos/memo.817
/home/sam/track/report
```

Professor:

Finding files based on type or size:

-type *fileType* File type matching f (file), d (directory), l (symbolic link)

-size *nSize* File size matches nSize where:
+n greater than n
-n less than n
n exactly n
Size is:
c - characters
k - kilobytes
M - megabytes
G - gigabytes

Finding files based on times:

-mtime *nTime* File modification time matching nTime:
+n more than n days ago
-n less than n days ago
n exactly n days ago

-newer *fileName* If the file being evaluated was modified more recently than the file named filename.

Note: When find figures out how many 24-hour periods ago the file was last accessed, any fractional part is ignored. So to match -atime +1, a file has to have been accessed at least two days ago.

Applying Multiple Criterion with find:

If we list multiple criterion, "and" is assumed by default. If we want to "or" the criteria, use "-o" between the criterion.

Find Actions:

By default, find prints the files that are found. We can also send the files directly to a command.

```
find dirList criteria -exec command {}\;
```

For each file matching the criteria, the file is sent to the command. The "{}" are replaced with the file by the find command.

Examples:

```
./sp1 -> Pgm1/p1
./Pgm1:
      cs2123p1Driver.c cs2123p1.h massign p1abc123.c p1Extra.txt p1OutExtra.txt
      cs2123p1Driver.o Makefile p1 p1abc123.o p1Input.txt p1Out.txt
./Pgm2:
      cs2123p2.docx cs2123p2Driver.o Makefile p2abc123.c p2Course.txt
      cs2123p2Driver.c cs2123p2.h p2 p2abc123.o p2Query.txt
./Pgm5:
      cs2123p5.h deleteCourse.c insert.o p5Driver.c p6Input.txt
      degreePlan.c deleteCourse.o Makefile p5Driver.o printFunctions.c
      degreePlan.o insert.c p5 p5Input.txt printFunctions.o

#Why doesn't this work?
$ find . -name *.o
find: No match
#The shell provides a list of all .o files in the current directory,
not subdirectories, as command arguments to find.

$ find .-name "*.o"
./Pgm1/cs2123p1Driver.o
./Pgm1/p1abc123.o
./Pgm5/deleteCourse.o
./Pgm5/degreePlan.o
./Pgm5/printFunctions.o
./Pgm5/insert.o
./Pgm5/p5Driver.o
./Pgm2/p2abc123.o
./Pgm2/cs2123p2Driver.o
```

```
$ find . -type d
.
./Pgm1
./Pgm5
./Pgm2
$ find . -type d #fox servers
.
./courses
./courses/cs3423
./courses/cs3423/find
./courses/cs3423/find/Pgm1
./courses/cs3423/find/Pgm5
./courses/cs3423/find/Pgm2
./courses/cs3423/myconfig
./courses/cs3423/myconfig/vim
./courses/cs3423/Spring25
./courses/cs3423/Spring25/assign2
```

```
$ find . -size +20k
./courses/cs3423/myconfig/vim/sh.vim
./courses/cs3423/grep/Program4/cs1713p4Driver.c
./courses/cs3423/grep/Program5/cs1713p5Driver.c

$ find . -type l #find symbolic links
./sp1            #none on fox servers lol

$ find . -type f -size -400c #find files smaller than 400 characters
./.vimrc
./courses/cs3423/find/Pgm1/p1OutExtra.txt
etc . . .

$ find . -mtime +200 #find files that were modified more than 200 days ago
./working
./working/java
./.cshrc

$ find Pgm5 -newer Pgm5/p5Driver.c
Pgm5
Pgm5/p5
Pgm5/p5Driver.o
#find files in directory Pgm5 that were modified after Pgm5/p5Driver.c

$ find Pgm5 -newer Pgm5/p5Driver.c -type f #modify above to only
Pgm5/p5                                    #provide files
Pgm5/p5Driver.o

$ find . -name “*.h” -o -name “*.c”
./Pgm1/p1abc123.c
./Pgm1/cs2123p1Driver.c
./Pgm1/cs2123p1.h
./Pgm5/deleteCourse.c
./Pgm5/printFunctions.c
./Pgm5/p5Driver.c
./Pgm5/insert.c
./Pgm5/degreePlan.c
./Pgm5/cs2123p5.h
./Pgm2/cs2123p2.h
./Pgm2/p2abc123.c
./Pgm2/cs2123p2Driver.c
#we can also use: $ find . -name “*.[hc]”

$ find . -name “*.h” -o -name “*.c” -exec -ls -al {} \;
-rw------- 1 clark faculty 3403 Jan 6 2017 ./Pgm1/p1abc123.c
-rw------- 1 clark faculty 13030 Dec 19 2016 ./Pgm1/cs2123p1Driver.c
etc . . .
#find each .h or .c file and list the details using ls
```

grep

[global regular expression print] Search for text in files

Syntax:

grep [options] pattern [fileList]

grep [options] pattern

The grep utility searches one or more text files for the specified pattern, which can be a simple string or another form of regular expression. The fileList is a list of files that grep searches; The shell provides the files.

The pattern is a regular expression; You must quote regular expressions that contain special characters, spaces, or tabs. The fileList is a list of the path names of ordinary text files that grep searches. With the -r option, fileList can contain directories; grep searches the files in these directory hierarchies.

Without any options, grep sends lines that contain a match for pattern to standard output. When you specify more than one file on the command line, grep precedes each line it displays with the name of the file it came from, followed by a colon.

Major Options (You can only use 1 of 3 at a time):

-G (regular/default) Interprets pattern as a basic regular expression.

-E (extended) Interprets pattern as an extended regular expression.

-F (fixed) Interprets pattern as a fixed string of characters.

```
#-P also uses perl regex
#grep -E is the same as egrep
#grep -F is the same as fgrep
```

Other important options:

-r recurse to subdirectories

-c count occurrences in each file

-w matches as a whole word not as a substring of another word

-v inverts the selection; only shows non-matching lines

-n display the line numbers for any matched text

-h don’t print the file name

uniq -c counts the number of occurrences of each unique line

A string literal is still a pattern, just a pattern that matches one thing.

```
$ grep cat * #find text lines containing “cat”
file4:John bought a cat, but it was really a very smelly cat.
file4:He had hoped to get a fun cat.
file5:The weather was getting very bad.  It was raining cats and dogs.
file5:what a catastrophe.
file5:I was so tired that I needed a cat nap.
grep: Program2: Is a directory
grep: Program3: Is a directory
grep: Program4: Is a directory
grep: Program5: Is a directory
#‘grep -F “cat” *’ is also an option, but unnecessary.
#The warning from grep is due to the fact that grep doesn’t do
#anything with directories by default. Those are error messages.

$ grep -F “cat” * 2>/dev/null #shows error messages going away
file4:John bought a cat, but it was really a very smelly cat.
file4:He had hoped to get a fun cat.
file5:The weather was getting very bad.  It was raining cats and dogs.
file5:what a catastrophe.
file5:I was so tired that I needed a cat nap.
```

The ‘*’ is the list of files we’ll search. It’s the same as
$ **grep -F “cat” Program2 Program3 Program4** etc. . .

You only really need quotes when there’s embedded white space.
Similar to situations such as:

```
#!/bin/bash
count=1
for arg; do
  echo “$count: $arg”
  let count++
done
```

```
$ ~/echoargs.bash this is a test
1: this
2: is
3: a
4: test
$ ~/echoargs.bash “this is a test”
1: this is a test
```

```
$ grep -r cat *
Program5/cs1713p5Driver.c:    // to allocate a new node
Program5/cs1713p5Driver.c:          exitError("Memory allocation error", "");
Program5/cs1713p5Driver.c:    - If the token is larger than the szTokenparm,
we return a truncated value.
Program5/cs1713p5Driver.c:   iCopy = MAX_TOKEN_SIZE; // truncated size
Program5/cs1713p5.h:NodeT *allocateNodeT(Book book);
#Finds text lines containing “cat” in the grep directory & its
subdirectories. The -r option recurses to subdirectories.

#or grep -Frc cat * (could just be -rc as well)
$ grep -F -r -c “cat” *
cs1713p0.c:0
cs1713p0v2.c:0
file4:2
file5:3
file6:0
Program2/p2Book.txt:0
Program2/cs1713p2.h:0
and so on. . .

#make it more useful
#sort by default does it in a lexicographical order
#use colons as a delimiter and numerically sort by field 2
$ grep -rc cat * | sort -t”:” -k2 -n
cs1713p0.c:0
cs1713p0v2.c:0
file6:0
Program2/cs1713p2.h:0
. . .
Program5/cs1713p5.h:1
file4:2
file5:3
Program3/cs1713p3Driver.c:3
Program4/cs1713p4Driver.c:10
Program5/cs1713p5Driver.c:10

#reverse sort - descending
$ grep -rc cat * | sort -t”:” -k2 -n -r
Program5/cs1713p5Driver.c:10
Program4/cs1713p4Driver.c:10
Program3/cs1713p3Driver.c:3
file5:3
file4:2
Program5/cs1713p5.h:1
. . .
```

```
#any lines ending in a ‘:0’ delete them!
#’$’ after 0 means it’s end of the line
$ grep -rc cat * | sort -t”:” -k2 -n -r | sed ‘/:0$/d’
Program5/cs1713p5Driver.c:10
Program4/cs1713p4Driver.c:10
Program3/cs1713p3Driver.c:3
file5:3
file4:2
Program5/cs1713p5.h:1
Program4/cs1713p4.h:1
```

Whole word match - The beginning and ending of a string occur at a word boundary

Word boundaries:

1. beginning of a line
2. end of a line
3. adjacent to a non-word character

Word characters: alphanumeric + underscore -> [a-zA-Z0-9_]

Non-word characters: -> [^a-zA-Z0-9_] (Anything that isn’t a letter, number, underscore

```
$ grep -rw cat *
file4:John bought a cat, but it was really a very smelly cat.
file4:He had hoped to get a fun cat.
file5:I was so tired that I needed a cat nap.
#Fewer results because it’s only a whole word match
#The comma in ‘cat,’ is a non-word character
#Same as the period in ‘cat.’ This also includes the spaces.

#’-v’ in most utilities means “verbose” so it tells it to produce more
#output and what you’d usually use in a debugging situation.
#But in grep, it means inverted. It’ll show you all of the lines that
#didn’t have a whole word match.
$ grep -wv “cat” ./file5
The weather was getting very bad.  It was raining cats and dogs.
It was so bad that I stepped in a poodle.
what a catastrophe.
```

We did an inversion with a whole word match, and so we’re basically saying “show me a line without a whole word match”.
“cats” has an s, “catastrophe” has an ‘a’ after “cat”. The space before catastrophe is fine, since it’s a non-word match, but again the ‘a’ fails it.

```
#-(Fixed String Match)(Recurse into Subdirectories)(Whole Word
Match)(Show us the line numbers) [The pattern is the word cat] [The
target is everything in the current directory]
$ grep -Frwn “cat” *
file4:1:John bought a cat, but it was really a very smelly cat.
file4:2:He had hoped to get a fun cat.
file5:4:I was so tired that I needed a cat nap.
```

Again -F is really for meta characters, so for something such as “***” you definitely need -F.

```
#Shows the lines containing #include statements sorted by include file
name. Don’t show the filenames.
$ grep -rh “#include” * | sort
#include "cs1713p0.h"
#include "cs1713p0.h"
#include "cs1713p3.h"
#include "cs1713p4.h"
#include "cs1713p5.h"
#include <stdio.h>
#include <stdio.h>
#include <stdio.h>
#include <stdio.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdlib.h>
#include <stdlib.h>
#include <string.h>
#include <string.h>
#include <string.h>
```

```
$ grep -rh “#include” * | uniq -c
```

Continued Important Grep Options:

-f compare a file of patterns against the fileList

```
grep -f patternFile fileList
```

Each line in the patternFile represents a pattern to be matched. The patterns can be regular expressions.

Example:

```
$ echo “my cat has a dog” | grep -e cat -e dog
1: my cat has a dog
```

```
#Given we have a txt file containing:
printf
sscanf
strcpy
$ grep -f example17pat.txt cs1713p0.c
 // about the safety of scanf and printf
      int iScanfCnt;     // sscanf returns the number of successful inputs
      printf("%-10s %-20s %10s %10s %10s %10s\n"
iScanfCnt = sscanf(szInputBuffer, "%lf %lf %lf %6s %20[^\n]\n"
    printf("invalid input when reading student data, only %d valid values.\n"
    printf("\tdata is %s\n", szInputBuffer);
printf("%-10s %-20s %10.2f %10.2f %10.2f %10.2f\n"

#What if we just wanted to include actual function calls and not the
context in which they're being used as well?
#In this situation we just modified our txt file, adding open
parenthesis to the end of each line:
#printf(
#sscanf(
#strcpy(
For the new output:
 printf("%-10s %-20s %10s %10s %10s %10s\n"
      iScanfCnt = sscanf(szInputBuffer, "%lf %lf %lf %6s %20[^\n]\n"
    printf("invalid input when reading student data, only %d valid values.\n"
    printf("\tdata is %s\n", szInputBuffer);
 printf("%-10s %-20s %10.2f %10.2f %10.2f %10.2f\n"
```

Options Continued:

`-l` when matching, only list the file names in the result instead of also showing the text line.

`-d skip` don't recurse into subdirectories. The -d option is used to specify what to do when a directory is encountered. In this case, we specify that they should be skipped.

Example:

```
$ grep -l -f example17pat.txt cs1713p0.c
cs1713p0.c

$ grep -rl "printf(" *
cs1713p0.c
cs1713p0v2.c
example17pat.txt
Program3/cs1713p3Driver.c
Program4/cs1713p4Driver.c
Program5/cs1713p5Driver.c
```

```
$ grep -d skip cat *
file4:John bought a cat, but it was really a very smelly cat.
file4:He had hoped to get a fun cat.
file5:The weather was getting very bad.  It was raining cats and dogs.
file5:what a catastrophe.
file5:I was so tired that I needed a cat nap.
$ grep -d read cat *
#default option and gives you all including grep errors
$ grep -d recurse cat *
#just a much longer way of typing -r
$ grep -l -d skip cat *
file4
file5
```

Given:

```
#!/bin/bash
 for file in $(grep -d skip -l $1 *); do
   echo "$file:"
   cat <$file
 done
```

Alternative way:

```
    if [ “$#” -eq 0 ]; then
      echo “Usage: $0 <pattern>”
      exit 1
    fi

  for file in `grep -d skip -l $1 *` ; do
     echo “$file:”
     cat < “$file”
     echo #echo by itself does a blank line
  done
```

```
$ ./example18.bash cat
example18.bash:
#!/bin/bash
# Use grep to get a list of files that contain the first argument.
# Use cat to show the contents of those files.
for file in $(grep -d skip -l $1 *); do
    echo "$file:" #Because our file example18 includes “cat” then it’s
    cat <$file    #also included in the output.
    done
file4:
John bought a cat, but it was really a very smelly cat.
He had hoped to get a fun cat.
file5:
The weather was getting very bad.  It was raining cats and dogs.
It was so bad that I stepped in a poodle.
what a catastrophe.
I was so tired that I needed a cat nap.
```

Options for grep from book:

`-c (-count)`
Displays only the number of lines that contain a match in each file.

`-C n (-context=n)`
Displays **n** lines of context around each matching line.

`-f file (-file=file)`
Reads *file*, which contains one pattern per line, and finds lines in the input that match each of the patterns.

`-h (-no-filename)`
Doesn't display the filename at the beginning of each line when searching multiple files.

`-i (-ignore-case)`
Ignores case-sensitive situations; Causes lowercase letters in the pattern to match uppercase letters in the file, and vice versa.

`-l (-files-with-matches)`
Displays only the name of each file that contains >=1 matches. Despite containing more than one match, a filename will only be displayed once.

`-m n (-max-count=n)`
Stops reaching each file/standard input after displaying **n** lines containing matches.

`-n (-line-number)`
Precedes each line by its line number in the file.

`-q (-quiet/silent)`
Doesn't write anything to standard output; only sets the exit code.

`-r/-R (-recursive)`
Recursively descends directories in the fileList and processes files within these directories.

`-s (-no-messages)`
Doesn't display an error message if a file in the fileList doesn't exist/isn't readable.

`-v (-invert-match)`
Displays all lines that don't contain a match for a pattern (inverse).

`-w (-word-regexp)`
The pattern must match a whole word.

`-x (-line-regexp)`
The pattern matches whole lines only.

☆ The grep utility returns an exit status of:
0 - match found
1 - no match found
2 - the file is inaccessible OR syntax error in grep command.

Examples:
Given the files:

testa	testb	testc
aaabb	aaaaa	AAAAA
bbbcc	bbbbb	BBBBB
ff-ff	ccccc	CCCCC
cccdd	ddddd	DDDDD
dddaa		

```
$ grep bb testa
aaabb
bbbcc

$ grep -v bb testa #inverse
ff-ff
cccdd
dddaa

$ grep -n bb testa #number line
1:aaabb
2:bbbcc

$ grep bb *
testa:aaabb
testa:bbbcc
test:bbbbb

$ grep -w bb * #whole word option - no output because no ‘bb’ by itself

$ grep -i bb * #match both upper and lowercase letters
testa:aaabb
testa:bbbcc
testb:bbbbb
test:BBBBB

$ grep -c bb * #num of lines in each file that match
testa:2
testb:1
test:0
```

```
$ grep -n !^' testa
1:aaabb
2:bbbcc
3:ff-ff
4:cccdd
5:dddaa

$ grep -h '#include' *.c | sort | uniq -c
9 #include "buff.h"
2 #include "poly.h"
1 #include "screen.h"
6 #include "window.h"
2 #include "x2.h"
2 #include "x3.h"
2 #include <math.h>
3 #include <stdio.h>
#suppress filenames from output, sort all the lines that match #include
#output repeated lines only once, along with a count of the number of
#repititions of each repeated line in its input
```