FREE eBook

LEARNING

grep

Free unaffiliated eBook created from
Stack Overflow contributors.

#grep

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: grep

It is an unofficial and free grep ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official grep.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with grep

Remarks

grep prints lines that contain a match for a pattern within files.

grep can use regular expressions and has several options to improve the quality of the results.

References

- POSIX
- FreeBSD man page
- OpenBSD man page
- GNU grep online manual
- Illumos man page

Versions

POSIX grep

Version	Release Date
POSIX.2	1992-01-01
IEEE Std 1003.1-2001	2001-12-06
IEEE Std 1003.1, 2004 Edition	2004-01-01
IEEE Std 1003.1, 2013 Edition	2013-04-19
IEEE Std 1003.1, 2016 Edition	2016-09-30

Illumos/OpenSolaris grep

Version	Release Date
2005-06-14	2005-06-14
2005-09-06	2005-09-06

Version	Release Date
2012-03-30	2012-03-30
2012-09-17	2012-09-17
2013-05-14	2013-05-14

GNU grep

Version	Release Date
2.0	1996-10-01
2.2	1998-04-27
2.3	1999-02-14
2.4.1	2000-03-01
2.4.2	2000-03-09
2.4	1999-12-03
2.5.1	2004-10-29
2.5.1a	2004-11-19
2.5.3	2007-08-02
2.5.4	2009-02-09
2.5	2002-03-13
2.6.1	2010-03-25
2.6.2	2010-03-29
2.6.3	2010-04-02
2.6	2010-03-23
2.7	2010-09-20
2.8	2011-05-13
2.9	2011-06-21
2.10	2011-11-16

Version	Release Date
2.11	2012-03-02
2.12	2012-04-23
2.13	2012-07-04
2.14	2012-08-20
2.15	2013-10-26
2.16	2014-01-01
2.17	2014-02-17
2.18	2014-02-20
2.19	2014-05-22
2.20	2014-06-03
2.21	2014-11-23
2.22	2015-11-01
2.23	2016-02-04
2.24	2016-03-10
2.25	2016-04-21
2.26	2016-10-02
2.27	2016-12-06
2.28	2017-02-06

BSD grep / FreeGrep

Version	Release Date
OpenBSD 3.0	2001-12-01
OpenBSD 3.4	2003-11-01
OpenBSD 3.5	2004-05-01
OpenBSD 3.6	2004-11-01

Version	Release Date
OpenBSD 3.7	2005-05-19
OpenBSD 3.8	2005-11-01
OpenBSD 3.9	2006-05-01
OpenBSD 4.0	2006-11-01
OpenBSD 4.1	2007-05-01
OpenBSD 4.3	2008-05-01
OpenBSD 4.8	2010-11-01
OpenBSD 5.0	2011-11-01
OpenBSD 5.3	2013-05-01
OpenBSD 5.7	2015-05-01
OpenBSD 5.8	2015-10-18
OpenBSD 5.9	2016-03-29
NetBSD 2.0	2004-12-09
NetBSD 4.0	2007-12-19
NetBSD 6.0	2012-10-17
NetBSD 7.0	2015-09-25
FreeBSD 9.0	2012-01-02
FreeBSD 10.0	2014-01-16

Examples

Basic usage

Running the command:

```
grep sam someFile.txt
```

When `someFile.txt` contains:

```
fred 14 m foo
sam 68 m bar
```

```
christina 83 f baz
bob 22 m qux
Sam 41 m quux
```

Will produce this output:

```
sam 68 m bar
```

Ignore case

Given a file `sample`:

```
hello
Hello
HELLO_there
```

A normal `grep` for "hello" returns:

```
$ grep "hello" sample
hello
```

Using `-i` allows to ignore case and match any "hello":

```
$ grep -i "hello" sample
hello
Hello
HELLO_there
```

Match whole words

Given a file `sample`:

```
hello world
ahello here
hello_there
```

A normal `grep` for "hello" returns:

```
$ grep hello sample
hello world
ahello here
hello_there
```

Using `-w` allows to select those lines containing matches that form whole words:

```
$ grep -w hello sample
hello world
```

Find text within a given directory, recursively

Using GNU grep

```
grep -r 'pattern' <directory path>
```

To also list line numbers of matches use `-n` option

```
grep -rn 'pattern' <directory path>
```

To search only files with particular glob pattern

```
grep --include='*.txt' -r 'pattern' <directory path>
```

Exclude file patterns or directories

```
grep -R --exclude=*.log 'pattern' <directory path>
grep -R --exclude={*.log,*.class} 'pattern' <directory path>

grep -R --exclude-dir=tmp 'pattern' <directory path>
grep -R --exclude-dir={tmp,lib} 'pattern' <directory path>
```

Notes and other useful options

- `<directory path>` can be skipped if searching in current directory
- The `-R` options follows all symbolic links, unlike `-r` which follows symbolic links only if they are on the command line
- `-l` to only list matching files
- `-h` to suppress filename prefix
- `--color=auto` to highlight matched patterns
- `-m <num>` to specify maximum number of matches for each file input

POSIX workaround to search recursively

```
find <directory path> -type f -exec grep -l 'pattern' {} +
```

- Options like `-n` , `-l` , etc can be used as required
- If `{} +` is not supported, use `{} \;` instead
- See find documentation for more help on `find` command like how to include/exclude file types, directories etc

Prints only the matching part of the lines

```
echo "Prints only the matching part of the lines" | grep -o "matching"
# prints matching
```

Grep Context Control

Given a file Sample called movieslist.

```
Troy
Gladiator
Robin Hood
King Arthur
BraveHeart
The Last Samurai
```

Normal grep returns

```
grep "Gladiator" movieslist
Gladiator
```

Now,using grep to print the below or above lines of the file.

To print the below line

```
grep -A 1 Gladiator movieslist
Gladiator
Robin Hood
```

To print the above line

```
grep -B 1 Gladiator movieslist
Troy
Gladiator
```

To print both

```
grep -C 1 Gladiator movieslist
Troy
Gladiator
Robin Hood
```

Read Getting started with grep online: https://riptutorial.com/grep/topic/2198/getting-started-with-grep

Chapter 2: Context line control

Remarks

`-A`, `-B` and `-C` options are not available in POSIX (see the POSIX specifications for `grep`).

Examples

Print lines before and/or after matching pattern

Usually `grep` prints only matching lines. In the example below `seq 9` generates a list of numbers from 1 to 9, one per line, and `grep` prints a single matching line:

```
seq 9 | grep 5
# 5
```

The `-C n` option (or `--context=n` in long form) prints `n` lines before and after each matching line, in addition to the matching line itself:

```
seq 9 | grep -C 2 '5'
# 3
# 4
# 5
# 6
# 7
```

Naturally, fewer than `n` lines will be printed if end-of-file or beginning-of-file is reached.

If we want to print lines only before or only after, but not both, we can use `-B n` (`--before-context=n`) or `-A n` (`--after-context=n`):

```
seq 9 | grep -B 2 '5'
# 3
# 4
# 5

seq 9 | grep -A 2 '5'
# 5
# 6
# 7
```

Note these options are not available in POSIX (see the POSIX specifications for `grep`).

If the contexts of two or more matching lines overlap, then all the lines are printed together as one large context. In the example below, `5` is part of the context of both `3` and `7`:

```
seq 9 | grep -E --context=2 '3|7'
# 1
```

```
# 2
# 3
# 4
# 5
# 6
# 7
# 8
# 9
```

However, if the contexts do not overlap, they are printed out with a group separator line. By default this is double hyphen (`--`):

```
seq 9 | grep -E --context=2 '2|8'
# 1
# 2
# 3
# 4
# --
# 6
# 7
# 8
# 9
```

We can set a different group separator line using the `--group-separator=SEP` option, or suppress this line entirely by using the `--no-group-separator` option:

```
seq 9 | grep -E --context=0 --group-separator='****' '2|8'
# 2
# ****
# 8

seq 9 | grep -E --context=0 --group-separator='' '2|8'
# 2
#
# 8

seq 9 | grep -E --context=0 --no-group-separator '2|8'
# 2
# 8
```

Finally, if we choose the `-v` option to print non-matching lines, then context is provided around those lines instead:

```
seq 9 | grep -E -v '1|3|4|5|6|7|9'
# 2
# --
# 8

seq 9 | grep -E -v -C 1 '1|3|4|5|6|7|9'
# 1
# 2
# 3
# --
# 7
# 8
# 9
```

Read Context line control online: https://riptutorial.com/grep/topic/4152/context-line-control

Chapter 3: Difference between grep, egrep, fgrep, pgrep.

Introduction

grep, **egrep**, **fgrep**, **rgrep**, **pgrep** - are commands in Unix-like operating systems that print lines matching a pattern. The grep searches the named input *FILEs* for lines containing a match to the given *PATTERN*. By default, it prints the matching lines. In addition, the variant programs **egrep**, **fgrep**, and rgrep are the same as **grep -E**, **grep -F**, and **grep -r**, respectively. These variants are deprecated, but are provided for backward compatibility.

Syntax

- `grep [OPTIONS] PATTERN [FILE...]`
- `grep [OPTIONS] [-e PATTERN]... [-f FILE]... [FILE...]`

Parameters

Symbol	Details Basic Regular Expressions (BRE)
`^`	the circumflex is used to match the beginning of a line.
`$`	used to match the end of a line.
`.`	matches any character except a new line.
`[]`	matches single character inside the brackets. If there's a `^` inside, it would match anything but the characters in the bracket.
`\`	before any of the non-alphanumeric characters quotes them.
`*`	symbol matches the preceding character or subexpression zero, one or more times.
`\1`	backreferences 1-9 match the exact text by the corresponding group.
`\{m,n\}`	matches the preceding elements at least *m* and no more than *n* times.
`|`	`foo|bar` matches foo or bar.
`\?`	short for {0,1}
`\+`	(short for **{1,}**) match the preceding character or subexpression at most 1 time, or at least 1 time respectively.

Symbol	Details Basic Regular Expressions (BRE)
`\n`	matches a newline, `\t` matches a tab, etc.
`\w`	matches any word constituent and \W matches any character that isn't a word constituent.
`\<\>`	match the empty string only at the beginning or end of a word
`\b`	matches either and **\B** matches where \b doesn't.
Symbol	**Details Extended Regular Expressions (ERE)**
`^`	match only at the beginning
`$`	match only at the end of a line.
`.`	matches any character (or any character except a newline).
`[…]`	matches any one character listed inside the brackets (character set). Add an initial ^ and ranges work like in BRE (see above).
`(…)`	syntactic group, for use with * or \DIGIT replacements.
`|`	for alternation: `foo`|`bar` matches foo or bar.
`*`	matches the preceding character or subexpression a number of times: 0, 1 or more times
`+`	matches 1 or more times preceding character.
`?`	matches preceding characters 0 or 1 times.
`\`	Backslash quotes the next character if it is not alphanumeric.
`{m,n}`	matches the preceding character or subexpression between m and n times (missing from some implementations); n or m can be omitted, and `{m}` means exactly m

Remarks

fgrep stands for "Fixed-string Global Regular Expressions Print". **fgrep** is the same as **`grep -F`**. This commands is a faster grep and behaves as grep but does NOT recognize any regular expression meta-characters as being special. The search will complete faster because it only processes a simple string rather than a complex pattern.

---

pgrep is an acronym that stands for "Process-ID Global Regular Expressions Print". pgrep looks through the currently running processes and lists the process IDs which matches the selection

criteria to stdout. pgrep is handy when all you want to know is the process id integer of a process.

grep	egrep(grep -E)	fgrep(grep -F)	pgrep
Basic Regular Expressions (BRE)	Extended Regular Expressions (ERE)	Searches only strings	Searches process by name

For more information and reference use some of the following links:

What is the difference between grep, egrep and fgrep ? Unix&Linux StackExchange

Why does my regular expression work in X but not in Y? Unix&Linux StackExchange

What is the difference between grep, pgrep, egrep, fgrep? Superuser

Examples

grep with Basic Regular Expressions

```
$ grep root /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
```

egrep with Extended Regular Expressions

```
$ egrep '^(0|1)+ [a-zA-Z]+$' searchfile.txt
011 AaBBS
```

fgrep with no Regular expressions

```
$ fgrep "." .bashrc
# will match lines with a dot.
```

pgrep with name of process

```
$ pgrep python
1299
```

Read Difference between grep, egrep, fgrep, pgrep. online:
https://riptutorial.com/grep/topic/8936/difference-between-grep--egrep--fgrep--pgrep-

Chapter 4: Regular expressions

Examples

Regular expressions

The search pattern can also be a regular expression. Running:

```
grep '^[A-Z]' someFile.txt
```

When `someFile.txt` contains:

```
fred 14 m foo
sam 68 m bar
christina 83 f baz
bob 22 m qux
Sam 41 m quux
```

Will produce the output:

```
Sam 41 m quux
```

since this is the only line in `someFile.txt` starting with an upper case letter.

Look behind

Given the following file:

```
hello how are you
i am fine
let's go, you!
let's go, baby!
```

`grep` with look-behind allows to print only some parts:

```
$ grep -Po "(?<=let's go, ).*" file
you!
baby!
```

In this case, it matches what occurs after "let's go, ".

Read Regular expressions online: https://riptutorial.com/grep/topic/4183/regular-expressions

Credits

S. No	Chapters	Contributors
1	Getting started with grep	Batsu, Benjamin W., Community, David Pärsson, depperm, dingalapadum, fedorqui, Jerry Jeremiah, kdhp, mszymborski, Stuxnet78, Sundeep, UNagaswamy
2	Context line control	Benjamin W., fedorqui, ghostarbeiter
3	Difference between grep, egrep, fgrep, pgrep.	Bor
4	Regular expressions	David Pärsson, dingalapadum, fedorqui