grep Command: Usage and Options

Introduction to grep

grep stands for "Global Regular Expression Print"[78]. It locates lines that contain matches of regular expression(s) and may or may not highlight the match depending on terminal capabilities[78].

Basic Syntax:[78]

```
grep [OPTIONS] [PATTERN] [FILE(S)]
```

Important Points:[78] - The pattern is a regular expression - Regular expressions may contain special characters (like *) having special meanings to the shell - Single quotes are recommended for quoting the pattern - Quoting enables searching for patterns containing spaces

---

grep Command Usage Examples

Basic Search

Search for word "method":[78]

```
$ grep method textfile.txt
```

Output:

```
trial division and Pollard's rho method. This is followed by two
sub-exponential algorithms based upon Fermat's factoring method. Dixon's
method uses random squares, but illustrates the basic concepts of the
and its variants. As a representative of the square-root methods for solving
the DLP, the baby-step-giant-step method is explained. Next, I introduce the
index calculus method (ICM) as a general paradigm for solving the DLP.
```
```[78]

**Search with space in pattern (must quote):**[78]
```bash
$ grep 'method ' textfile.txt
```

Output:

```
method uses random squares, but illustrates the basic concepts of the
the DLP, the baby-step-giant-step method is explained. Next, I introduce the
index calculus method (ICM) as a general paradigm for solving the DLP.
```
```[78]

**Correct quoting for pattern with special characters:**[78]
```bash
$ grep 'method[ \.]' textfile.txt
```

Output:

```
trial division and Pollard's rho method. This is followed by two
sub-exponential algorithms based upon Fermat's factoring method. Dixon's
method uses random squares, but illustrates the basic concepts of the
the DLP, the baby-step-giant-step method is explained. Next, I introduce the
index calculus method (ICM) as a general paradigm for solving the DLP.
```[78]

Error when pattern not quoted properly:[78]
```bash
$ grep method[ \.] textfile.txt
```

Output:

```
grep: Invalid regular expression
```

---

grep Options

-e Option: Multiple Patterns

Syntax:[78]

```
grep -e pattern1 -e pattern2 file
```

Search for multiple patterns:[78]

```
$ grep -e 'method' -e 'algorithm' textfile.txt
```

Output:

```
This tutorial focuses on algorithms for factoring large composite integers
public-key algorithms for encryption, key exchange, and digital signatures.
These algorithms highlight the roles played by the apparent difficulty of
Two exponential-time integer-factoring algorithms are first covered:
trial division and Pollard's rho method. This is followed by two
sub-exponential algorithms based upon Fermat's factoring method. Dixon's
method uses random squares, but illustrates the basic concepts of the
and its variants. As a representative of the square-root methods for solving
the DLP, the baby-step-giant-step method is explained. Next, I introduce the
index calculus method (ICM) as a general paradigm for solving the DLP.
```[78]

Handling patterns starting with hyphen:[78]

Without `-e`, patterns starting with `-` are interpreted as options:
```bash
$ grep '-key' textfile.txt
```

Output:

```
grep: invalid option -- 'k'
Usage: grep [OPTION]... PATTERNS [FILE]...
Try 'grep --help' for more information.
```

With `-e` option:

```
$ grep -e '-key' textfile.txt
```

Output:

```
public-key algorithms for encryption, key exchange, and digital signatures.
public-key protocols.
```[78]

-v Option: Inverted Search

Syntax:[78]
```bash
grep -v pattern file
```

Shows lines NOT containing the pattern:[78]

```
$ grep -v '[A-Z]' textfile.txt
```

Output:

```
public-key algorithms for encryption, key exchange, and digital signatures.
solving the factoring and discrete-logarithm problems, for designing
public-key protocols.
method uses random squares, but illustrates the basic concepts of the
candidates for smoothness testing and of sieving.
for extension fields of characteristic two.
```[78]

-i Option: Case-Insensitive Search

Syntax:[78]
```bash
grep -i pattern file
```

Case-sensitive search:[78]

```
$ grep 'method' textfile.txt
```

Output:

```
trial division and Pollard's rho method. This is followed by two
sub-exponential algorithms based upon Fermat's factoring method. Dixon's
method uses random squares, but illustrates the basic concepts of the
and its variants. As a representative of the square-root methods for solving
```

```
the DLP, the baby-step-giant-step method is explained. Next, I introduce the
index calculus method (ICM) as a general paradigm for solving the DLP.
```

Case-insensitive search:[78]

```
$ grep -i 'method' textfile.txt
```

Output:

```
trial division and Pollard's rho method. This is followed by two
sub-exponential algorithms based upon Fermat's factoring method. Dixon's
method uses random squares, but illustrates the basic concepts of the
Quadratic Sieve Method (QSM) which brings the benefits of using small
and its variants. As a representative of the square-root methods for solving
the DLP, the baby-step-giant-step method is explained. Next, I introduce the
index calculus method (ICM) as a general paradigm for solving the DLP.
```[78]

Note: Additional match "Quadratic Sieve Method (QSM)" with capital M

-w Option: Word-Based Search

Syntax:[78]
```bash
grep -w pattern file
```

All lines containing uppercase letters:[78]

```
$ grep '[A-Z]' textfile.txt
```

Output:

```
Abstract
This tutorial focuses on algorithms for factoring large composite integers
and for computing discrete logarithms in large finite fields. In order to
make the exposition self-sufficient, I start with some common and popular
These algorithms highlight the roles played by the apparent difficulty of
Two exponential-time integer-factoring algorithms are first covered:
trial division and Pollard's rho method. This is followed by two
sub-exponential algorithms based upon Fermat's factoring method. Dixon's
relation-collection and the linear-algebra stages. Next, I introduce the
Quadratic Sieve Method (QSM) which brings the benefits of using small
As the third module, I formally define the discrete-logarithm problem (DLP)
and its variants. As a representative of the square-root methods for solving
the DLP, the baby-step-giant-step method is explained. Next, I introduce the
index calculus method (ICM) as a general paradigm for solving the DLP.
Various stages of the basic ICM are explained both for prime fields and
```

Only whole-word single-letter uppercase words:[78]

```
$ grep -w '[A-Z]' textfile.txt
```

Output:

```
make the exposition self-sufficient, I start with some common and popular
relation-collection and the linear-algebra stages. Next, I introduce the
As the third module, I formally define the discrete-logarithm problem (DLP)
the DLP, the baby-step-giant-step method is explained. Next, I introduce the
```[78]

-n Option: Print Line Numbers

Syntax:[78]
```bash
grep -n pattern file
```

Without line numbers:[78]

```
$ grep '[^a-zA-Z]$' textfile.txt
```

Output:

```
public-key algorithms for encryption, key exchange, and digital signatures.
public-key protocols.
Two exponential-time integer-factoring algorithms are first covered:
candidates for smoothness testing and of sieving.
As the third module, I formally define the discrete-logarithm problem (DLP)
index calculus method (ICM) as a general paradigm for solving the DLP.
for extension fields of characteristic two.
```

With line numbers:[78]

```
$ grep -n '[^a-zA-Z]$' textfile.txt
```

Output:

```
6:public-key algorithms for encryption, key exchange, and digital signatures.
9:public-key protocols.
11:Two exponential-time integer-factoring algorithms are first covered:
17:candidates for smoothness testing and of sieving.
19:As the third module, I formally define the discrete-logarithm problem (DLP)
22:index calculus method (ICM) as a general paradigm for solving the DLP.
24:for extension fields of characteristic two.
```[78]

-c Option: Count Matches

Syntax:[78]
```bash
grep -c pattern file
```

Counts matching lines only:[78]

```
$ grep -c '[^a-zA-Z]$' textfile.txt
```

Output:

```
7
```[78]

-r or -R Option: Recursive Search

Syntax:[78]
```bash
grep -r pattern directory/
grep -R pattern directory/
```

Search recursively in subdirectories:[78]

```
$ grep -r 'nodep' .
```

Output:

```
./libstaque/static/defs.h:typedef node *nodep;
./libstaque/static/stack.h:typedef nodep stack;
./libstaque/static/queue.h:
nodep front;
./libstaque/static/queue.h:
nodep back;
./libstaque/shared/defs.h:typedef node *nodep;
./libstaque/shared/stack.h:typedef nodep stack;
./libstaque/shared/queue.h:
nodep front;
./libstaque/shared/queue.h:
nodep back;
```[78]

-l Option: List Filenames Only

Syntax:[78]
```bash
grep -l pattern file(s)
grep -r -l pattern directory/
```

Print only filenames with matches:[78]

```
$ grep -r -l 'nodep' .
```

Output:

```
./libstaque/static/defs.h
./libstaque/static/stack.h
./libstaque/static/queue.h
./libstaque/shared/defs.h
```

```
./libstaque/shared/stack.h
./libstaque/shared/queue.h
```[78]

---

grep Options Summary Table

Option	Description	Example
`-e pattern`	Specify multiple patterns	`grep -e 'foo' -e 'bar' file`
`-v`	Invert match (non-matching lines)	`grep -v 'pattern' file`
`-i`	Case-insensitive matching	`grep -i 'Pattern' file`
`-w`	Match whole words only	`grep -w 'word' file`
`-n`	Show line numbers	`grep -n 'pattern' file`
`-c`	Count matching lines	`grep -c 'pattern' file`
`-l`	List filenames only	`grep -l 'pattern' *.txt`
`-r, -R`	Recursive search	`grep -r 'pattern' dir/`

Combining Options

Multiple options can be used together:

```bash
$ grep -n -c 'pattern' file
```

Counts lines and shows line numbers[78]

```
$ grep -r -l -i 'pattern' .
```

Searches recursively, lists files only, case-insensitive[78]

```
$ grep -v -c 'pattern' file
```

Counts lines that do NOT match[78]

```
$ grep -w -n 'word' file
```

Shows matching whole words with line numbers[78]

Important Notes from PDF

Pattern Quoting:[78] - Always quote patterns with single quotes to prevent shell expansion - Quoting is especially important when pattern contains spaces or special characters

Multiple Pattern Matching:[78] - Use `-e` option for each pattern when searching for multiple patterns - Without `-e`, patterns starting with `-` will be misinterpreted as options

File Names in Output:[78] - When searching multiple files with `-r`, grep shows filename before match - Use `-l` to show only filenames - Use `-h` (not in exercises) to suppress filename display

Line Counting vs Output:[78] - `-c` shows count only (no lines displayed) - `-n` shows both lines and their numbers - Both can be used together for different purposes