Regular Expressions and grep: Complete Reference Guide

1. Basic Regular Expressions

Introduction to Regular Expressions

Regular expressions are patterns used to match character combinations in strings[78]. They are the same as those introduced in formal language theory but with different constructs. Regular expressions are used by `less`, `grep`, `sed`, `awk`, shells, and many text editors like `vi` and `emacs`[78].

Key Points:[78] - Used by many Unix/Linux tools - Searches are made line by line - Newline character is not allowed in regular expressions - Used for pattern matching and text processing

Basic Metacharacters

1. The Dot (.) - Match Any Character The period `.` matches any single character except newline[78].

Pattern: `a.g`

Example file content:

```
arg
aeg
aig
abg
a3g
a g
```

Command:

```
$ grep 'a.g' file.txt
```

Output:

```
arg
aeg
aig
abg
a3g
a g
```

The pattern `a.g` matches any three-character sequence starting with 'a', followed by any character, and ending with 'g'[78].

2. Character Classes [...] Character classes match any single character from the specified set[78].

Basic Syntax: - `[Tt]` - matches upper- or lower-case T - `[AEIOU]` - matches any upper-case vowel - `[a-z]` - matches any lower-case letter - `[a-zA-Z0-9]` - matches any alphanumeric character

Examples:

```
$ grep '[Tt]he' file.txt
```

Matches both "The" and "the"

```
$ grep '[A-Z][a-z ][a-z ].' file.txt
```

Matches capital letter followed by lowercase letter or space, followed by lowercase letter or space, followed by any character[78].

Character Ranges: - `[a-z]` - lowercase letters - `[A-Z]` - uppercase letters - `[0-9]` - digits - `[a-zA-Z]` - all letters - `[a-zA-Z0-9]` - alphanumeric characters

3. Negated Character Classes [^...] Use `^` after `[` to negate the character class[78].

Examples:

```
$ grep '[^ ]' file.txt
```

Matches any non-space character[78]

```
$ grep '[^aeiouAEIOU]' file.txt
```

Matches any character other than vowels[78]

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

Matches any non-alphabetic character[78]

Complex Example:

```
$ grep '[^AEIOU][^a-zA-Z ].....[a-drt]' file.txt
```

- `[^AEIOU]` - not an uppercase vowel
- `[^a-zA-Z ]` - not a letter or space
- `.....` - exactly 5 characters
- `[a-drt]` - ends with a, b, c, d, r, or t[78]

4. The Asterisk (*) - Zero or More The `*` quantifier matches zero or more of the preceding character or group[78].

Examples:

```
$ grep '.*' file.txt
```

Matches any string (including empty)[78]

```
$ grep '...*' file.txt
```

Matches any non-empty string[78]

```
$ grep '[a-z]*' file.txt
```

Matches any sequence of lowercase letters (including empty sequence)[78]

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

Matches any sequence of non-alphabetic characters[78]

Important Note: Longest possible matches are reported, starting as early as possible[78].

Example:

```
$ grep '[A-Z][a-zA-Z ]*[^ ]' file.txt
```

- `[A-Z]` - starts with uppercase letter
- `[a-zA-Z ]*` - followed by zero or more letters or spaces
- `[^ ]` - ends with non-space character[78]

Anchors

5. Start of Line (^) and End of Line ($) Start of Line (^): Use ^ as the first symbol to match at the beginning of a line[78].

```
$ grep '^[A-Z][a-z]*' file.txt
```

Matches the first word of a line if it starts with a capital letter[78]

**End of Line () : ∗ ∗ *Use*‘ as the last symbol to match at the end of a line[78].

```
$ grep '[a-z]*$' file.txt
```

Matches the last word of a line if it ends with lowercase letters[78]

Combined Example:

```
$ grep '^[A-Za-z, ]*$' file.txt
```

Matches entire lines containing only letters, commas, and spaces[78]

Escaping Special Characters

To match special characters literally, escape them with backslash \[78]:

Special Characters to Escape: - `\.` - literal dot - `\[` - literal opening bracket - `\]` - literal closing bracket - `\*` - literal asterisk - `\^` - literal caret - `\$` - literal dollar sign - `\\` - literal backslash - `\/` - literal forward slash (used in substitution)

Note: Hyphen - need not be quoted[78]

Examples:

```
$ grep '[a-z]*\.' file.txt
```

Matches lowercase letters followed by a literal period[78]

```
$ grep '[a-z]*-[a-z-]*.*\.' file.txt
```

Matches pattern with hyphen and ending with literal period[78]

2. grep Command

Introduction to grep

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

Basic Syntax:

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

Key Points:[78] - The pattern is a regular expression - Regular expressions may contain characters having special meanings to shell - Preferable to quote the pattern with single quotes - Quoting enables searching for patterns containing spaces

Basic grep Usage

Simple Pattern Matching Test File (textfile.txt):[78]

```
1    Abstract
2
3    This tutorial focuses on algorithms for factoring large composite integers
4    and for computing discrete logarithms in large finite fields. In order to
5    make the exposition self-sufficient, I start with some common and popular
6    public-key algorithms for encryption, key exchange, and digital signatures.
7    These algorithms highlight the roles played by the apparent difficulty of
8    solving the factoring and discrete-logarithm problems, for designing
9    public-key protocols.
10
11   Two exponential-time integer-factoring algorithms are first covered:
12   trial division and Pollard's rho method. This is followed by two
13   sub-exponential algorithms based upon Fermat's factoring method. Dixon's
14   method uses random squares, but illustrates the basic concepts of the
15   relation-collection and the linear-algebra stages. Next, I introduce the
16   Quadratic Sieve Method (QSM) which brings the benefits of using small
17   candidates for smoothness testing and of sieving.
18
19   As the third module, I formally define the discrete-logarithm problem (DLP)
20   and its variants. As a representative of the square-root methods for solving
21   the DLP, the baby-step-giant-step method is explained. Next, I introduce the
22   index calculus method (ICM) as a general paradigm for solving the DLP.
23   Various stages of the basic ICM are explained both for prime fields and
24   for extension fields of characteristic two.
```

25

Basic Search:

```
$ 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]

Quoted Pattern with Space:
```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]

Regular Expression with Character Class:
```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]

Complex Pattern:
```bash
$ grep '[a-z]*-[a-z-]*.*\.' textfile.txt
```

Output:

```
public-key algorithms for encryption, key exchange, and digital signatures.
public-key protocols.
sub-exponential algorithms based upon Fermat's factoring method. Dixon's
relation-collection and the linear-algebra stages. Next, I introduce the
the DLP, the baby-step-giant-step method is explained. Next, I introduce the
```

```[78]

## 3. grep Options

### 3.1 Multiple Pattern Search (-e)

The `-e` option allows specifying multiple patterns[78].

**Usage:**
```bash
grep -e pattern1 -e pattern2 file
```

Example:

```
$ 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]

Patterns Starting with Hyphen:
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.
```[78]

Correct way:
```bash
$ grep -e '-key' textfile.txt
```

Output:

```
public-key algorithms for encryption, key exchange, and digital signatures.
```
```

public-key protocols.
```[78]

### 3.2 Inverted Search (-v)

The `-v` option prints lines that do NOT contain matches[78].

**Example - Lines without uppercase letters:**
```bash
$ 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]

3.3 Case-Insensitive Search (-i)

The `-i` option performs case-insensitive matching[78].

Normal case-sensitive search:
```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
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]

Case-insensitive search:
```bash
$ 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]

Notice the additional match: "Quadratic Sieve Method (QSM)" due to case-insensitive matching

3.4 Word-Based Search (-w)

The `-w` option matches whole words only[78].

Example - Lines containing uppercase letters:
```bash
$ 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
```[78]

Lines containing single-letter uppercase words:
```bash
$ 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]

3.5 Line Numbers and Counting

Print Line Numbers (-n)

The `-n` option prints line numbers before matched lines[78].

Example:
```bash
$ 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.
```[78]

With line numbers:
```bash
$ 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]

Count Matches Only (-c)

The `-c` option prints only the count of matching lines[78].

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

Output:

```
7
```[78]

3.6 Recursive Search (-r, -R)

The `-r` or `-R` options search recursively in subdirectories[78].

Example:
```bash
$ 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]

3.7 List Filenames Only (-l)

The `-l` option prints only the names of files that contain matches[78].

Example:
```bash
$ 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]

3.8 Show Only Matches (-o)

The `-o` option shows only the matched part of the line (not in PDF, commonly used).

Example:
```bash
$ echo "The quick brown fox jumps" | grep -o 'qu[a-z]*'
```

Output:

```
quick
```

3.9 Show Context Lines (-A, -B, -C)

Show lines after (-A), before (-B), or around (-C) matches (not in PDF, commonly used).

Examples:

```
$ grep -A 2 -B 1 'pattern' file.txt    # 1 line before, 2 lines after
$ grep -C 3 'pattern' file.txt         # 3 lines before and after
```

3.10 Quiet Mode (-q)

The `-q` option suppresses output, useful in scripts for testing existence.

Example:

```
$ grep -q 'pattern' file.txt
$ echo $?    # 0 if found, 1 if not found
```

3.11 Suppress Filename Display (-h)

When searching multiple files, `-h` suppresses filename display.

Example:

```
$ grep -h 'pattern' file1.txt file2.txt
```

3.12 Show Filename Only (-H)

Force showing filename even for single file.

Example:

```
$ grep -H 'pattern' file.txt
```

4. Pattern Matching with grep

4.1 Extended Regular Expressions (-E)

Use `-E` for extended regular expressions (egrep equivalent).

Additional Metacharacters: - `+` - one or more - `?` - zero or one - `|` - alternation (OR) - `()` - grouping - `{n}` - exactly n times - `{n,}` - n or more times - `{n,m}` - between n and m times

Examples:

```
# One or more digits
$ grep -E '[0-9]+' file.txt

# Optional 's' at end
$ grep -E 'algorithms?' file.txt

# Either 'method' or 'algorithm'
$ grep -E 'method|algorithm' file.txt

# Grouping with alternation
$ grep -E '(public|private)-key' file.txt

# Exactly 3 digits
$ grep -E '[0-9]{3}' file.txt

# 2 to 4 letters
$ grep -E '[a-z]{2,4}' file.txt

# 5 or more characters
$ grep -E '.{5,}' file.txt
```

4.2 Quantifiers with Braces {n,m}

Exact Count {n}:

```
$ grep -E '[a-z]{5}' file.txt        # exactly 5 lowercase letters
$ grep -E '[0-9]{3}' file.txt        # exactly 3 digits
```

Minimum Count {n,}:

```
$ grep -E '[a-z]{5,}' file.txt       # 5 or more lowercase letters
$ grep -E '[0-9]{3,}' file.txt       # 3 or more digits
```

Range {n,m}:

```
$ grep -E '[a-z]{3,7}' file.txt      # 3 to 7 lowercase letters
$ grep -E '[0-9]{2,4}' file.txt      # 2 to 4 digits
```

4.3 Advanced Pattern Examples

Email Pattern:

```
$ grep -E '[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}' file.txt
```

IP Address Pattern:

```
$ grep -E '[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}' file.txt
```

Phone Number Patterns:

```
$ grep -E '[0-9]{3}-[0-9]{3}-[0-9]{4}' file.txt     # 123-456-7890
$ grep -E '\([0-9]{3}\)[0-9]{3}-[0-9]{4}' file.txt  # (123)456-7890
```

URL Pattern:

```
$ grep -E 'https?://[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}' file.txt
```

C Function Declaration:

```
$ grep -E '^[a-zA-Z_][a-zA-Z0-9_]*[ ]+[a-zA-Z_][a-zA-Z0-9_]*\(' file.c
```

4.4 Multiple Patterns with File Input (-f)

Store patterns in a file and use `-f` option.

patterns.txt:

```
method
algorithm
^[A-Z]
[0-9]+
```

Command:

```
$ grep -f patterns.txt textfile.txt
```

4.5 Perl-Compatible Regular Expressions (-P)

Use `-P` for Perl-compatible regular expressions (if available).

Examples:

```
$ grep -P '\d+' file.txt              # digits (same as [0-9]+)
$ grep -P '\w+' file.txt              # word characters
$ grep -P '\s+' file.txt              # whitespace characters
$ grep -P '(?=.*method)(?=.*test)' file.txt  # lookahead assertions
```

4.6 Practical Examples from PDF Exercises

Finding printf (not fprintf or sprintf)

```
$ grep -E '\bprintf\b' program.c
```

or

```
$ grep -w 'printf' program.c
```

C Block Analysis Lines opening blocks but not closing:

```
$ grep '{.*[^}]$' program.c
```

or

```
$ grep '{' program.c | grep -v '}'
```

Lines closing blocks but not opening:

```
$ grep '}' program.c | grep -v '{'
```

Lines both opening and closing blocks:

```
$ grep '{.*}' program.c
```

Number Range Matching (500-5000) Lines not containing 876:

```
$ grep -v '876' foonums.txt
```

Lines not containing unlucky numbers (500-5000):

```
$ grep -vE '[5-9][0-9]{2,3}|[1-4][0-9]{3}|5000' foonums.txt
```

Lines containing unlucky numbers but not 876:

```
$ grep -E '[5-9][0-9]{2,3}|[1-4][0-9]{3}|5000' foonums.txt | grep -v '876'
```

File Permission Analysis Files with owner execute permission:

```
$ ls -l | grep '^...x'
```

Non-directory files with owner execute permission:

```
$ ls -l | grep '^-..x'
```

Files >= 1MB (assuming 6+ digits in size column):

```
$ ls -l | grep -E ' [0-9]{7,} '
```

/etc/passwd Analysis Users with 4-digit UIDs:

```
$ grep -E ':[0-9]{4}:' /etc/passwd
```

Users with bash shell (excluding rbash):

```
$ grep ':/bin/bash$' /etc/passwd
```

Summary of All grep Options

Option	Description	Example
`-e pattern`	Specify pattern (multiple allowed)	`grep -e 'foo' -e 'bar' file`
`-v`	Invert match (show non-matching lines)	`grep -v 'pattern' file`
`-i`	Case-insensitive matching	`grep -i 'Pattern' file`
`-w`	Match whole words only	`grep -w 'cat' file`
`-n`	Show line numbers	`grep -n 'pattern' file`

Option	Description	Example
`-c`	Count matching lines only	`grep -c 'pattern' file`
`-l`	List filenames with matches	`grep -l 'pattern' *.txt`
`-r, -R`	Recursive search	`grep -r 'pattern' directory/`
`-o`	Show only matched parts	`grep -o 'pattern' file`
`-A n`	Show n lines after match	`grep -A 3 'pattern' file`
`-B n`	Show n lines before match	`grep -B 3 'pattern' file`
`-C n`	Show n lines around match	`grep -C 3 'pattern' file`
`-q`	Quiet mode (no output)	`grep -q 'pattern' file`
`-h`	Suppress filename in output	`grep -h 'pattern' *.txt`
`-H`	Show filename even for single file	`grep -H 'pattern' file`
`-E`	Extended regex (egrep)	`grep -E 'pattern|other' file`
`-F`	Fixed strings (fgrep)	`grep -F 'literal.string' file`
`-P`	Perl-compatible regex	`grep -P '\d+' file`
`-f file`	Read patterns from file	`grep -f patterns.txt file`

Complete Regular Expression Reference

Basic Regular Expression Elements

Element	Meaning	Example	Matches
`.`	Any character	`a.c`	abc, axc, a3c
`*`	Zero or more of preceding	`ab*c`	ac, abc, abbc
`^`	Start of line	`^The`	The (at line start)
`$`	End of line	`end$`	end (at line end)
`[abc]`	Any of a, b, or c	`[aeiou]`	vowels
`[a-z]`	Range a through z	`[0-9]`	digits
`[^abc]`	Not a, b, or c	`[^0-9]`	non-digits
`\`	Escape character	`\.`	literal dot

Element	Meaning	Example	Matches

Extended Regular Expression Elements (with -E)

Element	Meaning	Example	Matches
`+`	One or more	`ab+c`	abc, abbc, abbbc
`?`	Zero or one	`ab?c`	ac, abc
`|`	Alternation (OR)	`cat|dog`	cat or dog
`()`	Grouping	`(ab)+`	ab, abab, ababab
`{n}`	Exactly n times	`a{3}`	aaa
`{n,}`	n or more times	`a{2,}`	aa, aaa, aaaa…
`{n,m}`	Between n and m	`a{2,4}`	aa, aaa, aaaa

This comprehensive guide covers all aspects of regular expressions and grep usage as found in the PDF, plus commonly used extensions and practical examples for real-world usage.