Introduction to grep

The Unix command grep

- Abbreviation of *Global Regular Expression Print*.
- Locates lines that contains matches of regular expression(s).
- May or may not highlight the match.
- Options of grep enable you to do a lot of tasks with the matched lines.
- You run grep as:

```
grep <OPTIONS> <PATTERN> <FILE(S)>
```

- The pattern is a regular expression.
- A regular expression may contain characters (like *) having special meanings to the shell, so it is preferable to quote the pattern.
- Single (forward) quotes are recommended.
- Quoting also enables you to search for patterns containing space.

The input file used in the examples

The file textfile.txt

```
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
```

Search examples

```
$ grep method textfile.txt
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.
$ grep 'method ' textfile.txt
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.
$ grep method[ \.] textfile.txt
grep: Invalid regular expression
$ grep 'method[ \.]' textfile.txt
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.
$ grep '[a-z]*-[a-z-]*.*\.' textfile.txt
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
$
```

Making searches for multiple patterns

- Use the option **-e** multiple times.

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

- This option also helps you specify patterns starting with **-**.

```
$ grep '-key' textfile.txt
grep: invalid option - 'k'
Usage: grep [OPTION]... PATTERNS [FILE]...
Try 'grep -help' for more information.
$ grep -e '-key' textfile.txt
public-key algorithms for encryption, key exchange, and digital signatures.
public-key protocols.
$
```

The inverted search

- The option **-v** prints the lines that do not contain matches.

Lines not containing upper-case letters

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

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.

$
```

Case-insensitive search using the option –i

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

Word-based search using the option –w

Lines containing upper-case letters

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

Lines containing single-letter upper-case words

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

Printing line numbers and counting

- Use the option **-n** to print the line numbers before the printed lines.

Lines ending with non-alphabetic letters

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

- Use the option **-c** to print only the number of lines.

```
$ grep -c '[^a-zA-Z]$' textfile.txt
7
$
```

Search recursively in subdirectories

- Use the option **-r** or **-R**.

Recursive search for nodep in the current directory (,)

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

- Use the option **-l** only to print the names of the files that match. This is valid without the flag **-r** or **-R** as well.

```
$ grep -r -l 'nodep' .
./libstaque/static/defs.h
./libstaque/static/stack.h
./libstaque/static/queue.h
./libstaque/shared/defs.h
./libstaque/shared/stack.h
./libstaque/shared/queue.h
$
```

Introduction to sed

The Unix command sed

- Abbreviation of stream editor.
- Processes the input line by line.
- Works only on the selected lines.
- Can do a limited set of work on the selected lines.
- Sends the output to stdout (unless you specify a file name).
- **Does not modify the input file(s).**
- Run sed as:

```
$ sed <OPTIONS> 'LINE-SEL-CRIT COMMAND(s) <ARG(S)>' <FILE(S)>
```

- If the command(s) is/are written in **COMMANDFILE**, run sed as:

```
$ sed <OPTIONS> -f COMMANDFILE <FILE(S)>
```

Selection of lines

- **Selection by line numbers**
  - Lines are numbered $1, 2, 3, \ldots$.
  - The last line has the special *number* $.

n	Select only line number n
m,n	Select only lines m through n (all intermediate lines are included)
m,$	Select from the m-th line to the end (last line)

- **Selection by regular expression**
  - Supply the regular expression within a pair of delimiters (usually `/`)
- **Examples**

`sed '20 ...'`	Select line 20 only.
`sed '5,10 ...'`	Select lines 5 through 10.
`sed '1,$ ...'`	Select all lines.
`sed '/^[A-Z]/ ...'`	Select only the lines starting with an upper-case letter.

- If you want to select multiple ranges of lines, use multiple commands.

Summary of sed commands

p	Print the selected lines
i\text	Insert text before each selected line.
a\text	Append text after each selected line.
c\text	Replace the selected lines by text.
d	Delete the selected lines (not from the input file).
s/pattern/newstring/options	Substitute pattern by newstring in the selected lines. Some options may be additionally supplied.
q	Quit after the selected lines.
r infile	Insert the content of infile after the selected lines.
w outfile	Write the selected lines to outfile.

Printing selected lines

- sed always prints the input lines (after modifications if any).
- The lines selected for printing are printed **twice**.
- Run sed with the –n option to stop displaying the input lines.

Print lines ending with non-alphabetic characters

```
$ sed '/[^a-zA-Z]$/ p' textfile.txt
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
public-key algorithms for encryption, key exchange, and digital signatures.
public-key algorithms for encryption, key exchange, and digital signatures.
These algorithms highlight the roles played by the apparent difficulty of
solving the factoring and discrete-logarithm problems, for designing
public-key protocols.
public-key protocols.

Two exponential-time integer-factoring algorithms are first covered:
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
...
$
```

Printing only the selected lines

```
$ sed -n '/[^a-zA-Z]$/ p' textfile.txt
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.
$ sed -n '15p' textfile.txt
relation-collection and the linear-algebra stages. Next, I introduce the
$ sed -n '15,20p' textfile.txt
relation-collection and the linear-algebra stages. Next, I introduce the
Quadratic Sieve Method (QSM) which brings the benefits of using small
candidates for smoothness testing and of sieving.

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
$
```

Inserting and appending

```
$ sed '3,5i\A new line' textfile.txt
Abstract

A new line
This tutorial focuses on algorithms for factoring large composite integers
A new line
and for computing discrete logarithms in large finite fields. In order to
A new line
make the exposition self-sufficient, I start with some common and popular
public-key algorithms for encryption, key exchange, and digital signatures.
These algorithms highlight the roles played by the apparent difficulty of
solving the factoring and discrete-logarithm problems, for designing
public-key protocols.
...
$ sed '3,5a\A new line' textfile.txt
Abstract

This tutorial focuses on algorithms for factoring large composite integers
A new line
and for computing discrete logarithms in large finite fields. In order to
A new line
make the exposition self-sufficient, I start with some common and popular
A new line
public-key algorithms for encryption, key exchange, and digital signatures.
These algorithms highlight the roles played by the apparent difficulty of
solving the factoring and discrete-logarithm problems, for designing
public-key protocols.
$
```

Inserting and appending (continued)

```
$ sed -n '3,5i\A new line' textfile.txt
A new line
A new line
A new line
$ sed '1,$a\ ' textfile.txt
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

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

These algorithms highlight the roles played by the apparent difficulty of

solving the factoring and discrete-logarithm problems, for designing

public-key protocols.

...
$
```

- Multiple lines can be added using `\n`.

Replacing selected lines

```
$ sed '3,8c\New line\nAnother new line' textfile.txt
Abstract

New line
Another new line
public-key protocols.

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
relation-collection and the linear-algebra stages. Next, I introduce the
Quadratic Sieve Method (QSM) which brings the benefits of using small
candidates for smoothness testing and of sieving.
...
$
```

Deleting selected lines

```
$ sed '3,8d' textfile.txt
Abstract

public-key protocols.

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
relation-collection and the linear-algebra stages. Next, I introduce the
Quadratic Sieve Method (QSM) which brings the benefits of using small
candidates for smoothness testing and of sieving.

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
for extension fields of characteristic two.

$
```

Pattern substitution

- The command is **`s/pattern/replacement string/options`**.
- Here, **`pattern`** is a regular expression.
- The **`replacement string`** may be a constant string.
- The matched string can be addressed by **`&`**.
- Some operations are permitted in some versions of Unix.

  \U Convert to upper case

  \L Convert to lower case

- One or more of the following three options may be supplied.

  p Print the replaced strings to stdout (overriding –n).

  g Substitution is made only once in each selected line (default behavior). This option (meaning global) lets all possible substitutions in each line.

  w outfile Write the substituted lines to outfile.

Pattern substitution examples

Replace – by --

```
$ sed '/-/ s/-/--/' textfile.txt
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
public--key algorithms for encryption, key exchange, and digital signatures.
These algorithms highlight the roles played by the apparent difficulty of
solving the factoring and discrete--logarithm problems, for designing
public--key protocols.

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
...
$ sed -n '/-/ s/-/--/' textfile.txt
$ sed -n '/-/ s/-/--/p' textfile.txt
make the exposition self--sufficient, I start with some common and popular
public--key algorithms for encryption, key exchange, and digital signatures.
solving the factoring and discrete--logarithm problems, for designing
public--key protocols.
Two exponential--time integer-factoring algorithms are first covered:
sub--exponential algorithms based upon Fermat's factoring method. Dixon's
relation--collection and the linear-algebra stages. Next, I introduce the
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
$
```

Pattern substitution examples (continued)

Global substitution

```
$ sed -n '/-/ s/-/--/pg' textfile.txt
make the exposition self--sufficient, I start with some common and popular
public--key algorithms for encryption, key exchange, and digital signatures.
solving the factoring and discrete--logarithm problems, for designing
public--key protocols.
Two exponential--time integer--factoring algorithms are first covered:
sub--exponential algorithms based upon Fermat's factoring method. Dixon's
relation--collection and the linear--algebra stages. Next, I introduce the
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
$
```

Writing matched lines to matched.txt

```
$ sed -n '/-/ s/-/--/gw matched.txt' textfile.txt
$ cat matched.txt
make the exposition self--sufficient, I start with some common and popular
public--key algorithms for encryption, key exchange, and digital signatures.
solving the factoring and discrete--logarithm problems, for designing
public--key protocols.
Two exponential--time integer--factoring algorithms are first covered:
sub--exponential algorithms based upon Fermat's factoring method. Dixon's
relation--collection and the linear--algebra stages. Next, I introduce the
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
$
```

Pattern substitution application

Converting file to upper case: Method 1

```
$ sed '1,$s/.*/\U&/' textfile.txt
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
PUBLIC-KEY ALGORITHMS FOR ENCRYPTION, KEY EXCHANGE, AND DIGITAL SIGNATURES.
THESE ALGORITHMS HIGHLIGHT THE ROLES PLAYED BY THE APPARENT DIFFICULTY OF
SOLVING THE FACTORING AND DISCRETE-LOGARITHM PROBLEMS, FOR DESIGNING
PUBLIC-KEY PROTOCOLS.

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
RELATION-COLLECTION AND THE LINEAR-ALGEBRA STAGES. NEXT, I INTRODUCE THE
QUADRATIC SIEVE METHOD (QSM) WHICH BRINGS THE BENEFITS OF USING SMALL
CANDIDATES FOR SMOOTHNESS TESTING AND OF SIEVING.

...
$
```

Pattern substitution application (continued)

Converting file to upper case: Method 2

```
$ sed '1,$s/[a-z]/\U&/g' textfile.txt
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
PUBLIC-KEY ALGORITHMS FOR ENCRYPTION, KEY EXCHANGE, AND DIGITAL SIGNATURES.
THESE ALGORITHMS HIGHLIGHT THE ROLES PLAYED BY THE APPARENT DIFFICULTY OF
SOLVING THE FACTORING AND DISCRETE-LOGARITHM PROBLEMS, FOR DESIGNING
PUBLIC-KEY PROTOCOLS.

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
...
$ sed -n '1,$s/[a-z]/\U&/gp' textfile.txt
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
PUBLIC-KEY ALGORITHMS FOR ENCRYPTION, KEY EXCHANGE, AND DIGITAL SIGNATURES.
THESE ALGORITHMS HIGHLIGHT THE ROLES PLAYED BY THE APPARENT DIFFICULTY OF
SOLVING THE FACTORING AND DISCRETE-LOGARITHM PROBLEMS, FOR DESIGNING
PUBLIC-KEY PROTOCOLS.
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
$
```

Premature quitting

Quit after Line 5

```
$ sed '5q' textfile.txt
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
$
```

Quit after a line ending with a non-alphabetic character

```
$ sed '/[^a-zA-Z]$/q' textfile.txt
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
public-key algorithms for encryption, key exchange, and digital signatures.
$
```

Inserting an input file

Insert infile.txt after every line containing a hyphen

```
$ cat infile.txt
***
*** A hyphen is detected in the above line
***
$ sed '/-/ r infile.txt' textfile.txt
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
***
*** A hyphen is detected in the above line
***
public-key algorithms for encryption, key exchange, and digital signatures.
***
*** A hyphen is detected in the above line
***
These algorithms highlight the roles played by the apparent difficulty of
solving the factoring and discrete-logarithm problems, for designing
***
*** A hyphen is detected in the above line
***
public-key protocols.
***
*** A hyphen is detected in the above line
***
...
$
```

Write selected lines to a file

Write all lines not ending with alphabetic letters to outfile.txt

```
$ sed -n '/[^a-zA-Z]$/ w outfile.txt' textfile.txt
$ cat outfile.txt
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.
$
```

Negating a command

- The negated command is denoted by a preceding **!**.
- The action in the command is taken on the **unselected** lines.

Print lines 1–15

```
$ sed -n '16,$!p' textfile.txt
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
public-key algorithms for encryption, key exchange, and digital signatures.
These algorithms highlight the roles played by the apparent difficulty of
solving the factoring and discrete-logarithm problems, for designing
public-key protocols.

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
relation-collection and the linear-algebra stages. Next, I introduce the
$
```

Negating a command (continued)

Print lines not containing –

```
$ sed -n '/-/!p' textfile.txt
Abstract

This tutorial focuses on algorithms for factoring large composite integers
and for computing discrete logarithms in large finite fields. In order to
These algorithms highlight the roles played by the apparent difficulty of

trial division and Pollard's rho method. This is followed by two
method uses random squares, but illustrates the basic concepts of the
Quadratic Sieve Method (QSM) which brings the benefits of using small
candidates for smoothness testing and of sieving.

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
for extension fields of characteristic two.

$
```

Negating a command (continued)

Add a new line after every non-empty line

```
$ sed '/^$/!a\ ' textfile.txt
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

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

These algorithms highlight the roles played by the apparent difficulty of

solving the factoring and discrete-logarithm problems, for designing

public-key protocols.


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

...
$
```

Running multiple commands

- Commands can be separated by semicolon (`;`).
- But `;` can be a part of a string (for example, during insert, append, or change).
- A better option is to supply each command with the option **`-e`**.
- The commands are executed in the sequence in which they appear.
- The order of the commands may matter.
- Another option is to write multiple commands (possibly with different selection criteria), and invoke sed with the **`-f`** option.

Multiple commands separated by ;

```
$ sed '3,4p;8p;9q' textfile.txt
Abstract

This tutorial focuses on algorithms for factoring large composite integers
This tutorial focuses on algorithms for factoring large composite integers
and for computing discrete logarithms in large finite fields. In order to
and for computing discrete logarithms in large finite fields. In order to
make the exposition self-sufficient, I start with some common and popular
public-key algorithms for encryption, key exchange, and digital signatures.
These algorithms highlight the roles played by the apparent difficulty of
solving the factoring and discrete-logarithm problems, for designing
solving the factoring and discrete-logarithm problems, for designing
public-key protocols.
$
```

Problems with the separator approach

Wrong identification of a command as text

```
$ sed '/[A-Z]/a\Substitution made above;1,$s/[A-Z]/\L&/g' textfile.txt
Abstract
Substitution made above;1,$s/[A-Z]/L&/g

This tutorial focuses on algorithms for factoring large composite integers
Substitution made above;1,$s/[A-Z]/L&/g
and for computing discrete logarithms in large finite fields. In order to
Substitution made above;1,$s/[A-Z]/L&/g
make the exposition self-sufficient, I start with some common and popular
Substitution made above;1,$s/[A-Z]/L&/g
public-key algorithms for encryption, key exchange, and digital signatures.
...
$
```

Getting it done by multiple –e options

```
$ sed -e '/[A-Z]/a\Substitution made above' -e '1,$s/[A-Z]/\L&/g' textfile.txt
abstract
Substitution made above

this tutorial focuses on algorithms for factoring large composite integers
Substitution made above
and for computing discrete logarithms in large finite fields. in order to
Substitution made above
make the exposition self-sufficient, i start with some common and popular
Substitution made above
public-key algorithms for encryption, key exchange, and digital signatures.
these algorithms highlight the roles played by the apparent difficulty of
Substitution made above
solving the factoring and discrete-logarithm problems, for designing
public-key protocols.

two exponential-time integer-factoring algorithms are first covered:
Substitution made above
...
$
```

Note: Swapping the two commands (first substitute, then append) does not append at all...

The order matters

. . . because after global substitution, no line contains any upper-case letter.

First substitute, then quit

```
$ sed -e '/-/s/-/--/g' -e '/-/q' textfile.txt
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
$
```

First quit (then substitute)

```
$ sed -e '/-/q' -e '/-/s/-/--/g' textfile.txt
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
$
```

Writing sed commands in a file

```
$ cat commands.txt
/[A-Z]/a\Substitution made above
1,$s/[A-Z]/\L&/g
$ sed -f commands.txt textfile.txt
abstract
Substitution made above

this tutorial focuses on algorithms for factoring large composite integers
Substitution made above
and for computing discrete logarithms in large finite fields. in order to
Substitution made above
make the exposition self-sufficient, i start with some common and popular
Substitution made above
public-key algorithms for encryption, key exchange, and digital signatures.
these algorithms highlight the roles played by the apparent difficulty of
Substitution made above
solving the factoring and discrete-logarithm problems, for designing
public-key protocols.

two exponential-time integer-factoring algorithms are first covered:
Substitution made above
...
$
```

Writing an executable sed script

```
$ which sed
/usr/bin/sed
$ cat commands.sed
#!/usr/bin/sed -f
/[A-Z]/a\Substitution made above
1,$s/[A-Z]/\L&/g
$ chmod 755 commands.sed
$ ./commands.sed textfile.txt
abstract
Substitution made above

this tutorial focuses on algorithms for factoring large composite integers
Substitution made above
and for computing discrete logarithms in large finite fields. in order to
Substitution made above
make the exposition self-sufficient, i start with some common and popular
Substitution made above
public-key algorithms for encryption, key exchange, and digital signatures.
these algorithms highlight the roles played by the apparent difficulty of
Substitution made above
solving the factoring and discrete-logarithm problems, for designing
public-key protocols.

two exponential-time integer-factoring algorithms are first covered:
Substitution made above
...
$
```