A VISUAL REFERENCE · sed

sed,
one change at a time

Transform text by following one input line, one command and one output at a time. One small input. One focused change. Its actual result.

The canonical Org file owns the programs and synthetic fixtures. Each complete standalone Bash script recreates its exact inputs in an owned temporary directory. The final PDF is versioned at the repository root.

> **Last changes: 2026-11-01.** First edition: 96 deliberately small examples, complete inputs, authored expected outcomes, BSD/GNU comparisons and a separate GNU-extension chapter.

96
worked examples

16
topic families

1
canonical Org source

READING THE TOUR

Follow one change

Read the input

All files consumed by a command are shown completely. Small inputs make every selected, changed or omitted line visible.

Change one feature

Follow the comparison link. Read the focused command and its named switch or instruction before adding another feature.

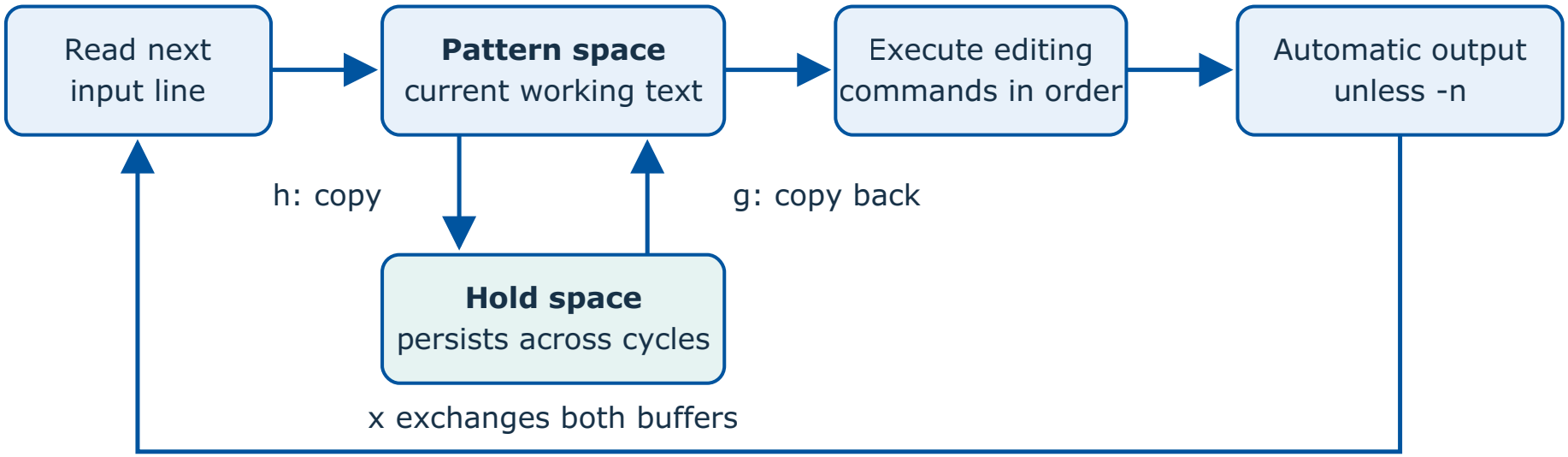
Check the actual result

Stdout and the complete driver status are checked against independently authored expectations. Empty stdout is displayed explicitly.

The gallery, search, topic filter and stable SED IDs support quick lookup. Copy the focused command while using the shown fixture, or download the complete script to recreate that fixture. The complete script runs with LC_ALL=C in a fresh directory and removes that owned directory on exit. In file-editing examples, only those reconstructed local copies are changed.

The normal sed cycle

The diagram shows the ordinary cycle. Pattern space is the current working text; hold space is a separate, initially empty buffer that persists between cycles. h and g copy in opposite directions; x exchanges the buffers. Flow commands such as d, D, n, N and q change this normal sequence and are introduced individually after the basic printing and editing operations.



Implementations and reproducibility

The common examples are tested with macOS BSD sed and GNU sed. Common means supported by those tested implementations, not that every option is specified by POSIX. The Implementation row identifies the GNU-only examples; they explicitly use gsed. Their complete scripts select gsed when installed, or provide that command name using an ordinary sed executable verified as GNU. The focused GNU commands can use the ordinary name on a GNU/Linux installation. Versions and binary identities participate in the execution fingerprint.

The outer captured status belongs to the Bash driver. Expected failures are caught and their tool statuses are printed explicitly. Implementation-specific diagnostic prose is not a golden output. No stdout means no output bytes, not a missing execution. All fixtures are original invented teaching data.

Related tours and primary references

- Regular-expression tour: BRE, ERE, captures and matching operations
- Bash tour: quoting, pipelines, redirections and exit statuses
- grep: selecting and inspecting text
- GNU sed manual
- POSIX sed specification

The examples, explanations, fixtures and expected outcomes are independently authored. The references establish command and dialect semantics; implementation extensions are labeled separately.

01 - Understand normal output

Start with three lines. Separate automatic printing, explicit printing and suppression before changing any text.

SED-01-001

Copy input with an empty program

The empty program changes nothing. Automatic printing still emits each input line.

Input

data/letters.txt

```
red
blue
green
```

Code

```
sed '' data/letters.txt
```

Actual output

Captured run · exit 0

```
red
blue
green
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`''`	Run no editing command; keep the normal output cycle.

Download: SED-01-001.sh

SED-01-002

Suppress automatic printing

Add -n to the same empty program. A successful run now emits zero bytes.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-01-001 · Copy input with an empty program.

Code

```
sed -n '' data/letters.txt
```

Actual output

Captured run · exit 0

```
(no stdout)
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`-n`	Suppress automatic printing, not input reading.

Download: SED-01-002.sh

SED-01-003

Print explicitly while automatic output is active

The p command and normal printing both run, so each line appears twice.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-01-001 · Copy input with an empty program.

Code

```
sed 'p' data/letters.txt
```

Actual output

Captured run · exit 0

```
red
red
blue
blue
green
green
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`p`	Print the current pattern space immediately.

Download: SED-01-003.sh

SED-01-004

Use one explicit output path

Add -n to p, leaving only the explicit output.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-01-003 · Print explicitly while automatic output is active.

Code

```
sed -n 'p' data/letters.txt
```

Actual output

Captured run · exit 0

```
red
blue
green
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`-n p`	Suppress the automatic copy and keep the explicit one.

Download: SED-01-004.sh

SED-01-005

Pass a program with -e

Use an explicit program argument before adding more programs later.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-01-004 · Use one explicit output path.

Code

```
sed -e 's/red/RED/' data/letters.txt
```

Actual output

Captured run · exit 0

```
RED
blue
green
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`-e`	Supply one sed program on the command line.

Download: SED-01-005.sh

SED-01-006

Read the same program from a file

The script fixture contains the exact substitution used in the preceding example.

Input

data/edit.sed

```
s/red/RED/
```

data/letters.txt

```
red
blue
green
```

Code

```
sed -f data/edit.sed data/letters.txt
```

Actual output

Captured run · exit 0

```
RED
blue
green
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`-f`	Read editing instructions from the named script file.

Download: SED-01-006.sh

02 - Address lines by number

An address chooses where an instruction runs; it does not itself change or remove text.

SED-02-001

Print the first input line

An address restricts the explicit print instruction.

Input

data/lines.txt

```
one
two
three
four
five
six
```

Code

```
sed -n '1p' data/lines.txt
```

Actual output

Captured run · exit 0

```
one
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`1`	Run p only at the first line.

Download: SED-02-001.sh

SED-02-002

Print the second input line

Change only the address number.

Input

data/lines.txt

```
one
two
three
four
five
six
```

Compare with SED-02-001 · Print the first input line.

Code

```
sed -n '2p' data/lines.txt
```

Actual output

Captured run · exit 0

```
two
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`2`	Select a different single line.

Download: SED-02-002.sh

SED-02-003

Print the final input line

The final-line address avoids needing the input length in advance.

Input

data/lines.txt

```
one
two
three
four
five
six
```

Compare with SED-02-002 · Print the second input line.

Code

```
sed -n '$p' data/lines.txt
```

Actual output

Captured run · exit 0

```
six
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`$`	Address the last input line.

Download: SED-02-003.sh

SED-02-004

Print an inclusive numeric range

Both endpoint lines are included.

Input

data/lines.txt

```
one
two
three
four
five
six
```

Compare with SED-02-003 · Print the final input line.

Code

```
sed -n '2,4p' data/lines.txt
```

Actual output

Captured run · exit 0

```
two
three
four
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`2,4`	Select lines two through four inclusively.

Download: SED-02-004.sh

SED-02-005

Continue the range through the end

Replace the numeric endpoint with the final-line address.

Input

data/lines.txt

```
one
two
three
four
five
six
```

Compare with SED-02-004 · Print an inclusive numeric range.

Code

```
sed -n '2,$p' data/lines.txt
```

Actual output

Captured run · exit 0

```
two
three
four
five
six
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`2,$`	Select from the second line through the final input line.

Download: SED-02-005.sh

SED-02-006

Invert a single address

The exclamation applies the instruction everywhere outside that address.

Input

data/lines.txt

```
one
two
three
four
five
six
```

Compare with SED-02-002 · Print the second input line.

Code

```
sed -n '2!p' data/lines.txt
```

Actual output

Captured run · exit 0

```
one
three
four
five
six
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`!`	Negate the selected address.

Download: SED-02-006.sh

03 - Address lines by content

Use a regex address before p, then expand the selected region one boundary at a time.

SED-03-001

Print lines matching an address

The regex selects where p runs; it is not itself a substitution.

Input

data/log.txt

```
INFO start
WARN retry
ERROR disk
INFO ready
```

Code

```
sed -n '/WARN/p' data/log.txt
```

Actual output

Captured run · exit 0

```
WARN retry
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`/regex/`	Address lines containing the regex.

Download: SED-03-001.sh

SED-03-002

Anchor the address at the start

Make the expected log-level position explicit.

Input

data/log.txt

```
INFO start
WARN retry
ERROR disk
INFO ready
```

Compare with SED-03-001 · Print lines matching an address.

Code

```
sed -n '/^WARN/p' data/log.txt
```

Actual output

Captured run · exit 0

```
WARN retry
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`^`	Require the matching level at the beginning of the line.

Download: SED-03-002.sh

SED-03-003

Print a delimited inclusive region

Both marker lines and the content between them are selected.

Input

data/block.txt

```
before
BEGIN
red
blue
END
after
```

Code

```
sed -n '/^BEGIN$/,/^END$/p' data/block.txt
```

Actual output

Captured run · exit 0

```
BEGIN
red
blue
END
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`Two regex addresses`	Enter at BEGIN and include the first later END.

Download: SED-03-003.sh

SED-03-004

Continue from a marker to the input end

Change only the ending address of the preceding region.

Input

data/block.txt

```
before
BEGIN
red
blue
END
after
```

Compare with SED-03-003 · Print a delimited inclusive region.

Code

```
sed -n '/^BEGIN$/,$p' data/block.txt
```

Actual output

Captured run · exit 0

```
BEGIN
red
blue
END
after
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`/BEGIN/,$`	Keep all lines from the starting marker onward.

Download: SED-03-004.sh

SED-03-005

Inspect a range whose first line matches its end regex

For this two-address range, the ending regex is tested from the next line, so the first matching line does not immediately close the range.

Input

data/starts.txt

```
START
middle
START
end
```

Code

```
sed -n '1,/^START$/p' data/starts.txt
```

Actual output

Captured run · exit 0

```
START
middle
START
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`1,/regex/`	Observe when the second address begins being tested.

Download: SED-03-005.sh

SED-03-006

Print the complement of an addressed set

Address negation applies to complete lines, just as with a numeric address.

Input

data/log.txt

```
INFO start
WARN retry
ERROR disk
INFO ready
```

Compare with SED-03-001 · Print lines matching an address.

Code

```
sed -n '/^INFO/!p' data/log.txt
```

Actual output

Captured run · exit 0

```
WARN retry
ERROR disk
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`/regex/!`	Run p only when the line does not match the address.

Download: SED-03-006.sh

04 - Make small substitutions

Separate the pattern, replacement, delimiter and occurrence selector. All initial transformations write to stdout.

SED-04-001

Replace the first occurrence on a line

A substitution without an occurrence flag changes only the first match in each pattern space.

Input

data/repeated.txt

```
red red red
blue blue
```

Code

```
sed 's/red/RED/' data/repeated.txt
```

Actual output

Captured run · exit 0

```
RED red red
blue blue
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`s/old/new/`	Replace the first matching substring.

Download: SED-04-001.sh

SED-04-002

Replace every nonoverlapping occurrence

Add only the global flag.

Input

data/repeated.txt

```
red red red
blue blue
```

Compare with SED-04-001 · Replace the first occurrence on a line.

Code

```
sed 's/red/RED/g' data/repeated.txt
```

Actual output

Captured run · exit 0

```
RED RED RED
blue blue
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`g`	Repeat the substitution across this line.

Download: SED-04-002.sh

SED-04-003

Replace only the second occurrence

Use an occurrence number instead of g.

Input

data/repeated.txt

```
red red red
blue blue
```

Compare with SED-04-002 · Replace every nonoverlapping occurrence.

Code

```
sed 's/red/RED/2' data/repeated.txt
```

Actual output

Captured run · exit 0

```
red RED red
blue blue
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`2`	Replace the second matching occurrence on each line.

Download: SED-04-003.sh

SED-04-004

Replace a match with an empty string

An empty replacement removes the matched text, not the whole input line.

Input

data/repeated.txt

```
red red red
blue blue
```

Compare with SED-04-001 · Replace the first occurrence on a line.

Code

```
sed 's/red//' data/repeated.txt
```

Actual output

Captured run · exit 0

```
 red red
blue blue
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`Empty replacement`	Remove only the first matched substring.

Download: SED-04-004.sh

SED-04-005

Choose a delimiter that fits the data

Pipe delimiters make the path slashes ordinary pattern characters.

Input

data/paths.txt

```
old=/srv/data
new=/srv/cache
```

Code

```
sed 's|/srv/data|/opt/data|' data/paths.txt
```

Actual output

Captured run · exit 0

```
old=/opt/data
new=/srv/cache
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`Alternate delimiter`	Keep slash-heavy strings readable.

Download: SED-04-005.sh

SED-04-006

Write the same change with slash delimiters

Only the delimiter escaping changes; the result stays identical.

Input

data/paths.txt

```
old=/srv/data
new=/srv/cache
```

Compare with SED-04-005 · Choose a delimiter that fits the data.

Code

```
sed 's/\/srv\/data/\/opt\/data/' data/paths.txt
```

Actual output

Captured run · exit 0

```
old=/opt/data
new=/srv/cache
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`\/`	Escape a slash when slash is the substitution delimiter.

Download: SED-04-006.sh

05 - Choose the regex notation

Use BRE first and switch to ERE explicitly. A shell string and a sed regex are separate syntax layers.

SED-05-001

Observe a wildcard period

An unescaped period matches one character in the middle of both records.

Input

data/dots.txt

```
a.b
aXb
```

Code

```
sed 's/a.b/MATCH/' data/dots.txt
```

Actual output

Captured run · exit 0

```
MATCH
MATCH
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`.`	Match one character rather than a literal period.

Download: SED-05-001.sh

SED-05-002

Require a literal period

A bracket expression restricts the substitution to literal punctuation.

Input

data/dots.txt

```
a.b
aXb
```

Compare with SED-05-001 · Observe a wildcard period.

Code

```
sed 's/a[.]b/MATCH/' data/dots.txt
```

Actual output

Captured run · exit 0

```
MATCH
aXb
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`[.]`	Treat the period as data.

Download: SED-05-002.sh

SED-05-003

Choose characters with a class

Select c-words with either a or o in the vowel position.

Input

data/animals.txt

```
cat
cot
cut
dog
```

Code

```
sed 's/c[ao]t/PET/' data/animals.txt
```

Actual output

Captured run · exit 0

```
PET
PET
cut
dog
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`[ao]`	Accept a or o at that one position.

Download: SED-05-003.sh

SED-05-004

Repeat a BRE atom

The star belongs to the preceding bracket expression and consumes the complete digit run.

Input

data/ids.txt

```
ID-7
ID-42
ID-123
```

Code

```
sed 's/[0-9][0-9]*/#/' data/ids.txt
```

Actual output

Captured run · exit 0

```
ID-#
ID-#
ID-#
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`[0-9][0-9]*`	Require one ASCII digit and then zero or more additional digits.

Download: SED-05-004.sh

SED-05-005

Use a BRE interval

Backslashed braces express an exact count in BRE.

Input

data/ids.txt

```
ID-7
ID-42
ID-123
```

Compare with SED-05-004 · Repeat a BRE atom.

Code

```
sed 's/[0-9]\{3\}/#/' data/ids.txt
```

Actual output

Captured run · exit 0

```
ID-7
ID-42
ID-#
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`\{3\}`	Require three digits in the matched substring.

Download: SED-05-005.sh

SED-05-006

Switch to ERE for the same interval

Add -E and remove the BRE brace escapes. The selected text stays the same.

Input

data/ids.txt

```
ID-7
ID-42
ID-123
```

Compare with SED-05-005 · Use a BRE interval.

Code

```
sed -E 's/[0-9]{3}/#/' data/ids.txt
```

Actual output

Captured run · exit 0

```
ID-7
ID-42
ID-#
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`-E`	Select extended regex syntax explicitly.

Download: SED-05-006.sh

06 - Reuse matched and captured text

The replacement has its own meaning for ampersand and backreferences. Keep captured digit strings intact.

SED-06-001

Insert the whole match into replacement text

Ampersand expands to the complete matched name assignment.

Input

data/values.txt

```
name=alice
name=bob
port=8080
```

Code

```
sed -n 's/^name=.*/[&]/p' data/values.txt
```

Actual output

Captured run · exit 0

```
[name=alice]
[name=bob]
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`&`	Insert the whole match in the replacement.

Download: SED-06-001.sh

SED-06-002

Insert a literal ampersand

Escape the replacement ampersand when it should not expand.

Input

data/values.txt

```
name=alice
name=bob
port=8080
```

Compare with SED-06-001 · Insert the whole match into replacement text.

Code

```
sed -n 's/^name=.*/A\&B/p' data/values.txt
```

Actual output

Captured run · exit 0

```
A&B
A&B
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`\&`	Write a literal ampersand in replacement text.

Download: SED-06-002.sh

SED-06-003

Capture a name using BRE groups

Backslashed parentheses capture the part after name=.

Input

data/values.txt

```
name=alice
name=bob
port=8080
```

Compare with SED-06-002 · Insert a literal ampersand.

Code

```
sed -n 's/^name=\([a-z][a-z]*\)$/user:\1/p' data/values.txt
```

Actual output

Captured run · exit 0

```
user:alice
user:bob
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`\(...\) and \1`	Capture a name and insert the first captured string.

Download: SED-06-003.sh

SED-06-004

Use ERE groups for the same capture

Change the grouping and repetition notation along with -E.

Input

data/values.txt

```
name=alice
name=bob
port=8080
```

Compare with SED-06-003 · Capture a name using BRE groups.

Code

```
sed -En 's/^name=([a-z]+)$/user:\1/p' data/values.txt
```

Actual output

Captured run · exit 0

```
user:alice
user:bob
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`ERE groups`	Keep replacement backreference syntax while changing the pattern dialect.

Download: SED-06-004.sh

SED-06-005

Reorder two captured fields

The replacement puts the captured digit string first.

Input

data/names.txt

```
alice:17
bob:204
```

Code

```
sed -En 's/^([a-z]+):([0-9]+)$/\2 \1/p' data/names.txt
```

Actual output

Captured run · exit 0

```
17 alice
204 bob
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`\2 then \1`	Insert captures in a different order.

Download: SED-06-005.sh

SED-06-006

Reuse the same capture twice

A replacement can insert captured text several times without matching the input again.

Input

data/names.txt

```
alice:17
bob:204
```

Compare with SED-06-005 · Reorder two captured fields.

Code

```
sed -En 's/^([a-z]+):([0-9]+)$/\1=\2; owner=\1/p' data/names.txt
```

Actual output

Captured run · exit 0

```
alice=17; owner=alice
bob=204; owner=bob
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`Repeated backreference`	Reuse an already captured string in the replacement.

Download: SED-06-006.sh

07 - Combine addresses and substitution flags

Decide where a substitution may run independently from which successful results are printed.

SED-07-001

Print only successful substitutions

Suppress automatic output and use the substitution p flag.

Input

data/settings.txt

```
# title
mode=old
note=old
# end
```

Code

```
sed -n 's/old/NEW/p' data/settings.txt
```

Actual output

Captured run · exit 0

```
mode=NEW
note=NEW
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`s///p`	Print a line only when that substitution succeeds.

Download: SED-07-001.sh

SED-07-002

See p and automatic printing together

Remove only -n. Successful lines now appear both explicitly and automatically.

Input

data/settings.txt

```
# title
mode=old
note=old
# end
```

Compare with SED-07-001 · Print only successful substitutions.

Code

```
sed 's/old/NEW/p' data/settings.txt
```

Actual output

Captured run · exit 0

```
# title
mode=NEW
mode=NEW
note=NEW
note=NEW
# end
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`Automatic plus explicit output`	Do not interpret duplicated output as two input records.

Download: SED-07-002.sh

SED-07-003

Combine global substitution with conditional printing

-n suppresses unchanged lines, while g changes all occurrences before p prints the result.

Input

data/repeated.txt

```
red red red
blue blue
```

Compare with SED-04-002 · Replace every nonoverlapping occurrence.

Code

```
sed -n 's/red/RED/gp' data/repeated.txt
```

Actual output

Captured run · exit 0

```
RED RED RED
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`gp`	Change all occurrences and print a successful result.

Download: SED-07-003.sh

SED-07-004

Print only if a second occurrence exists

The occurrence selector controls whether substitution succeeds at all.

Input

data/repeated.txt

```
red red red
blue blue
```

Compare with SED-07-003 · Combine global substitution with conditional printing.

Code

```
sed -n 's/red/RED/2p' data/repeated.txt
```

Actual output

Captured run · exit 0

```
red RED red
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`2p`	Replace the second match and print only on success.

Download: SED-07-004.sh

SED-07-005

Restrict substitution with a regex address

The address selects the record; the substitution then changes its value.

Input

data/settings.txt

```
# title
mode=old
note=old
# end
```

Code

```
sed '/^mode=/s/old/NEW/' data/settings.txt
```

Actual output

Captured run · exit 0

```
# title
mode=NEW
note=old
# end
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`Address before s`	Keep the edit confined to a selected record.

Download: SED-07-005.sh

SED-07-006

Restrict substitution with a numeric address

Use the equivalent known line position for this fixed fixture.

Input

data/settings.txt

```
# title
mode=old
note=old
# end
```

Compare with SED-07-005 · Restrict substitution with a regex address.

Code

```
sed '2s/old/NEW/' data/settings.txt
```

Actual output

Captured run · exit 0

```
# title
mode=NEW
note=old
# end
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`2s`	Apply the same edit only to line two.

Download: SED-07-006.sh

08 - Delete, insert and append lines

Delete ends the current cycle; inserted or appended text is a separate output operation.

SED-08-001

Delete one addressed line

d ends this cycle without automatic printing of the deleted line.

Input

data/letters.txt

```
red
blue
green
```

Code

```
sed '2d' data/letters.txt
```

Actual output

Captured run · exit 0

```
red
green
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`d`	Discard this pattern space and advance input.

Download: SED-08-001.sh

SED-08-002

Delete an inclusive range

Expand the selected region without changing the deletion instruction.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-08-001 · Delete one addressed line.

Code

```
sed '2,3d' data/letters.txt
```

Actual output

Captured run · exit 0

```
red
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`2,3d`	Discard both selected input lines.

Download: SED-08-002.sh

SED-08-003

Delete only empty lines

An empty-line expression does not match the line containing three spaces.

Input

data/blanks.txt

```
first


last
```

Code

```
sed '/^$/d' data/blanks.txt
```

Actual output

Captured run · exit 0

```
first

last
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`^$`	Recognize a zero-character line.

Download: SED-08-003.sh

SED-08-004

Delete whitespace-only lines too

Widen the address while keeping the same deletion command.

Input

data/blanks.txt

```
first


last
```

Compare with SED-08-003 · Delete only empty lines.

Code

```
sed '/^[[:space:]]*$/d' data/blanks.txt
```

Actual output

Captured run · exit 0

```
first
last
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`Whitespace class`	Recognize empty and whitespace-only lines.

Download: SED-08-004.sh

SED-08-005

Insert text before a selected line

The portable text-command form uses a backslash followed by a literal program newline.

Input

data/letters.txt

```
red
blue
green
```

Code

```
sed '2i\
extra
' data/letters.txt
```

Actual output

Captured run · exit 0

```
red
extra
blue
green
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`i followed by text`	Write inserted text before the addressed input line.

Download: SED-08-005.sh

SED-08-006

Append text after a selected line

Change only the text command from i to a; the inserted text stays the same.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-08-005 · Insert text before a selected line.

Code

```
sed '2a\
extra
' data/letters.txt
```

Actual output

Captured run · exit 0

```
red
blue
extra
green
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`a followed by text`	Queue text after the addressed input line.

Download: SED-08-006.sh

09 - Change text, translate characters and quit

These commands operate at different levels: complete selected lines, individual characters, line labels and input flow.

SED-09-001

Replace one complete line

c replaces the entire selected line rather than matching a substring inside it.

Input

data/letters.txt

```
red
blue
green
```

Code

```
sed '2c\
replacement
' data/letters.txt
```

Actual output

Captured run · exit 0

```
red
replacement
green
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`c followed by text`	Change the complete addressed input line.

Download: SED-09-001.sh

SED-09-002

Replace a whole addressed range once

The common range form emits one replacement when the selected range ends.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-09-001 · Replace one complete line.

Code

```
sed '2,3c\
replacement
' data/letters.txt
```

Actual output

Captured run · exit 0

```
red
replacement
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`Range c`	Collapse the addressed region to one replacement text.

Download: SED-09-002.sh

SED-09-003

Translate one character everywhere

y maps characters position by position; it does not interpret a regex.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-09-002 · Replace a whole addressed range once.

Code

```
sed 'y/r/R/' data/letters.txt
```

Actual output

Captured run · exit 0

```
Red
blue
gReen
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`y/from/to/`	Replace every r character with R.

Download: SED-09-003.sh

SED-09-004

Print input line numbers

The equals command emits the current line number; -n suppresses the normal line text.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-09-003 · Translate one character everywhere.

Code

```
sed -n '=' data/letters.txt
```

Actual output

Captured run · exit 0

```
1
2
3
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`=`	Print a one-based input line number.

Download: SED-09-004.sh

SED-09-005

Quit after a selected line

q stops before reading later lines. Normal printing still emits the current selected line.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-09-004 · Print input line numbers.

Code

```
sed '2q' data/letters.txt
```

Actual output

Captured run · exit 0

```
red
blue
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`q`	Stop after this cycle instead of processing the rest of the stream.

Download: SED-09-005.sh

SED-09-006

Inspect pattern space with visible line endings

The l command lists the current pattern space and marks its end with a dollar sign. That marker is display output, not part of the input.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-09-005 · Quit after a selected line.

Code

```
sed -n 'l' data/letters.txt
```

Actual output

Captured run · exit 0

```
red$
blue$
green$
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`l`	Make line endings and escaped characters visible for inspection.

Download: SED-09-006.sh

10 - Scope a group of instructions

Braces let several commands share an address. The commands still run in their written order.

SED-10-001

Give one command a regex scope

The group and its address determine where the enclosed edit runs.

Input

data/log.txt

```
INFO start
WARN retry
ERROR disk
INFO ready
```

Code

```
sed '/^WARN/{s/WARN/NOTE/;}' data/log.txt
```

Actual output

Captured run · exit 0

```
INFO start
NOTE retry
ERROR disk
INFO ready
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`{ ... }`	Run a scoped command group.

Download: SED-10-001.sh

SED-10-002

Add a second instruction to that scope

Both changes are confined to the WARN record and run in written order.

Input

data/log.txt

```
INFO start
WARN retry
ERROR disk
INFO ready
```

Compare with SED-10-001 · Give one command a regex scope.

Code

```
sed '/^WARN/{s/WARN/NOTE/;s/retry/again/;}' data/log.txt
```

Actual output

Captured run · exit 0

```
INFO start
NOTE again
ERROR disk
INFO ready
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`Two scoped instructions`	Apply the second edit to the already modified pattern space.

Download: SED-10-002.sh

SED-10-003

Negate the scope of a group

Only records outside the INFO address receive a prefix.

Input

data/log.txt

```
INFO start
WARN retry
ERROR disk
INFO ready
```

Compare with SED-10-002 · Add a second instruction to that scope.

Code

```
sed '/^INFO/!{s/^/[other] /;}' data/log.txt
```

Actual output

Captured run · exit 0

```
INFO start
[other] WARN retry
[other] ERROR disk
INFO ready
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`Address ! group`	Apply the complete group only to nonmatching records.

Download: SED-10-003.sh

SED-10-004

Share a numeric range across a group

One range selects where both substitutions may run.

Input

data/letters.txt

```
red
blue
green
```

Code

```
sed '2,3{s/^/[selected] /;s/$/!/;}' data/letters.txt
```

Actual output

Captured run · exit 0

```
red
[selected] blue!
[selected] green!
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`Range before braces`	Give several instructions the same inclusive line range.

Download: SED-10-004.sh

SED-10-005

Print and then end the selected cycle

Explicit p emits the addressed line; d prevents a second automatic copy.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-10-004 · Share a numeric range across a group.

Code

```
sed '2{p;d;}' data/letters.txt
```

Actual output

Captured run · exit 0

```
red
blue
green
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`p then d`	Emit once and skip the rest of this cycle.

Download: SED-10-005.sh

SED-10-006

Observe that deletion skips later instructions

After d, the replacement command in the same group cannot run on that line.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-10-005 · Print and then end the selected cycle.

Code

```
sed '2{d;s/.*/unreachable/;}' data/letters.txt
```

Actual output

Captured run · exit 0

```
red
green
```

Parameter	Meaning in this example
Implementation	BSD/GNU common
Cycle termination	d immediately advances to a fresh input cycle.

Download: SED-10-006.sh

11 - Move text through hold space

Pattern space holds the current working text. Hold space starts empty and persists between cycles. Lowercase h/g replace a buffer; uppercase H/G append with a newline; x exchanges both.

SED-11-001

Save a line and retrieve it later

h copies the first line to hold space. g on the third line replaces that pattern space with the saved text.

Input

data/letters.txt

```
red
blue
green
```

Code

```
sed -n '1h;3g;p' data/letters.txt
```

Actual output

Captured run · exit 0

```
red
blue
red
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`h and g`	Hold keeps red across later input cycles.

Download: SED-11-001.sh

SED-11-002

Retrieve without changing the saved buffer

Add g on the second line too. Both later retrievals read the same saved red.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-11-001 · Save a line and retrieve it later.

Code

```
sed -n '1h;2g;3g;p' data/letters.txt
```

Actual output

Captured run · exit 0

```
red
red
red
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`g`	Replace pattern space; leave hold space unchanged.

Download: SED-11-002.sh

SED-11-003

Exchange pattern space and hold space

Change the second-line g to x. That exchanges red and blue, so the final g now retrieves blue.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-11-002 · Retrieve without changing the saved buffer.

Code

```
sed -n '1h;2x;3g;p' data/letters.txt
```

Actual output

Captured run · exit 0

```
red
red
blue
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`x`	After line two: pattern=red, hold=blue.

Download: SED-11-003.sh

SED-11-004

Append a line to hold space

H keeps the saved text and appends a newline plus the current line. The final retrieval therefore prints two lines.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-11-003 · Exchange pattern space and hold space.

Code

```
sed -n '1h;2H;3g;p' data/letters.txt
```

Actual output

Captured run · exit 0

```
red
blue
red
blue
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`H`	After line two: hold contains red, newline, blue.

Download: SED-11-004.sh

SED-11-005

Append hold space to the current pattern

Change only H to G. G appends to pattern space, leaving the saved red available for the third-line g.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-11-004 · Append a line to hold space.

Code

```
sed -n '1h;2G;3g;p' data/letters.txt
```

Actual output

Captured run · exit 0

```
red
blue
red
red
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`G`	The second pattern becomes blue, newline, red; hold stays red.

Download: SED-11-005.sh

SED-11-006

Retrieve the initially empty hold space

Before h or H has stored anything, hold space is empty. Add 1g to the earlier l example and observe the first empty record as a visible dollar marker.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-09-006 · Inspect pattern space with visible line endings.

Code

```
sed -n '1g;l' data/letters.txt
```

Actual output

Captured run · exit 0

```
$
blue$
green$
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`Initial hold state`	Distinguish an empty pattern space from zero output bytes.

Download: SED-11-006.sh

12 - Read and print multiline pattern space

n and N advance input differently. p prints all of pattern space; P prints through its first embedded newline; D restarts after dropping the first part.

SED-12-001

Advance with lowercase n

Without -n, n prints the current pattern space, reads the next line and resumes the program there.

Input

data/pairs.txt

```
a1
a2
b1
b2
```

Code

```
sed 'n' data/pairs.txt
```

Actual output

Captured run · exit 0

```
a1
a2
b1
b2
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`n`	Start the next input line within the current program pass.

Download: SED-12-001.sh

SED-12-002

Print the line read by n explicitly

With -n, n does not print the previous pattern. The following p prints each second line.

Input

data/pairs.txt

```
a1
a2
b1
b2
```

Compare with SED-12-001 · Advance with lowercase n.

Code

```
sed -n 'n;p' data/pairs.txt
```

Actual output

Captured run · exit 0

```
a2
b2
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`-n with n`	Separate advancing input from deciding what to output.

Download: SED-12-002.sh

SED-12-003

Append the next line with uppercase N

N retains the current pattern and appends a newline plus the next input line. p prints each resulting pair.

Input

data/pairs.txt

```
a1
a2
b1
b2
```

Compare with SED-12-002 · Print the line read by n explicitly.

Code

```
sed -n 'N;p' data/pairs.txt
```

Actual output

Captured run · exit 0

```
a1
a2
b1
b2
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`N`	Build a two-line pattern space without an intermediate output.

Download: SED-12-003.sh

SED-12-004

Print only the first part with P

Change p to P. The second line remains in pattern space but is not emitted by P.

Input

data/pairs.txt

```
a1
a2
b1
b2
```

Compare with SED-12-003 · Append the next line with uppercase N.

Code

```
sed -n 'N;P' data/pairs.txt
```

Actual output

Captured run · exit 0

```
a1
b1
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`P`	Print through the first embedded newline only.

Download: SED-12-004.sh

SED-12-005

Restart after dropping the first part with D

D keeps the remaining part and restarts the program. The final lone line reaches N at EOF under -n and emits nothing.

Input

data/pairs.txt

```
a1
a2
b1
b2
```

Compare with SED-12-004 · Print only the first part with P.

Code

```
sed -n 'N;P;D' data/pairs.txt
```

Actual output

Captured run · exit 0

```
a1
a2
b1
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`D`	Drop the first part and restart without reading a fresh cycle immediately.

Download: SED-12-005.sh

SED-12-006

Join each complete pair with a space

Replace the embedded newline produced by N. The fixture deliberately contains two complete pairs.

Input

data/pairs.txt

```
a1
a2
b1
b2
```

Compare with SED-12-003 · Append the next line with uppercase N.

Code

```
sed 'N;s/\n/ /' data/pairs.txt
```

Actual output

Captured run · exit 0

```
a1 a2
b1 b2
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`\n in the pattern`	Match the embedded newline in a multiline pattern space.

Download: SED-12-006.sh

13 - Branch after a transformation

Labels and branches control which instruction runs next. Every loop here makes progress on a tiny fixed input. Use program newlines after branch commands and labels; a semicolon after b is not a portable BSD/GNU command separator.

SED-13-001

Branch to the end of a program

An unlabelled b skips the remaining instructions; normal end-of-cycle output remains active. End b with a program newline for BSD/GNU compatibility.

Input

data/letters.txt

```
red
blue
green
```

Code

```
sed 'b
s/red/RED/' data/letters.txt
```

Actual output

Captured run · exit 0

```
red
blue
green
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`b`	Skip to the end of this program pass.

Download: SED-13-001.sh

SED-13-002

Branch only on a selected record

Other records can still reach the substitution after the addressed branch.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-13-001 · Branch to the end of a program.

Code

```
sed '/^blue$/b
s/r/R/g' data/letters.txt
```

Actual output

Captured run · exit 0

```
Red
blue
gReen
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`Addressed b`	Skip the rest of this pass only on the selected blue line.

Download: SED-13-002.sh

SED-13-003

Jump to a named label

The labelled branch skips one edit but continues at the following labelled section.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-13-002 · Branch only on a selected record.

Code

```
sed 'b done
s/red/RED/
:done
s/blue/BLUE/' data/letters.txt
```

Actual output

Captured run · exit 0

```
red
BLUE
green
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`:label`	Name a later program location for b to jump to.

Download: SED-13-003.sh

SED-13-004

Branch after a successful substitution

t branches only if a substitution succeeded since input was read or the preceding t was evaluated.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-13-003 · Jump to a named label.

Code

```
sed 's/red/RED/
t done
s/blue/BLUE/
:done' data/letters.txt
```

Actual output

Captured run · exit 0

```
RED
BLUE
green
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`t`	Test whether a preceding substitution succeeded.

Download: SED-13-004.sh

SED-13-005

Skip a fallback edit after success

A selected replacement branches past the fallback; all other records receive the fallback prefix.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-13-004 · Branch after a successful substitution.

Code

```
sed 's/red/RED/
t done
s/^/other: /
:done' data/letters.txt
```

Actual output

Captured run · exit 0

```
RED
other: blue
other: green
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`Conditional fallback`	Keep successful and fallback transformations separate.

Download: SED-13-005.sh

SED-13-006

Repeat a substitution while it makes progress

Each successful replacement shortens a run of spaces. t returns to the label until no double space remains.

Input

data/spaces.txt

```
a  b   c
keep
```

Code

```
sed ':again
s/  / /
t again' data/spaces.txt
```

Actual output

Captured run · exit 0

```
a b c
keep
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`Progressing loop`	Every iteration removes one extra space from this finite input.

Download: SED-13-006.sh

14 - Read scripts and write owned files

All paths are inside the temporary directory created by the complete script. Original repository inputs remain unchanged; backups are displayed explicitly.

SED-14-001

Preview an edit before writing a file

Normal sed output leaves the input copy unchanged. Display it afterwards to show the distinction.

Input

data/letters.txt

```
red
blue
green
```

Code

```
sed 's/red/RED/' data/letters.txt
printf 'input remains:\n'
cat data/letters.txt
```

Actual output

Captured run · exit 0

```
RED
blue
green
input remains:
red
blue
green
```

Parameter	Meaning in this example
Implementation	BSD/GNU common
Stdout preview	Transform output without editing even the local input copy.

Download: SED-14-001.sh

SED-14-002

Save transformed output to another file

Shell redirection writes a different path in the owned directory.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-14-001 · Preview an edit before writing a file.

Code

```
sed 's/red/RED/' data/letters.txt > edited.txt
cat edited.txt
```

Actual output

Captured run · exit 0

```
RED
blue
green
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`> OTHER_FILE`	Write a distinct output path instead of overwriting the file being read.

Download: SED-14-002.sh

SED-14-003

Read extra text after an addressed line

The r command queues the named file content after the selected line.

Input

data/banner.txt

```
NOTICE
```

data/letters.txt

```
red
blue
green
```

Code

```
sed '2r data/banner.txt' data/letters.txt
```

Actual output

Captured run · exit 0

```
red
blue
NOTICE
green
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`r FILE`	Append the external text at the addressed output position.

Download: SED-14-003.sh

SED-14-004

Write selected pattern spaces to a file

-n suppresses normal stdout; the following cat displays the file written by w.

Input

data/letters.txt

```
red
blue
green
```

Code

```
sed -n '/^blue$/w selected.txt' data/letters.txt
cat selected.txt
```

Actual output

Captured run · exit 0

```
blue
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`w FILE`	Write selected pattern spaces to the named owned output file.

Download: SED-14-004.sh

SED-14-005

Write only successfully changed lines

The substitution w flag records successful results while -n suppresses normal output.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-14-004 · Write selected pattern spaces to a file.

Code

```
sed -n 's/red/RED/w changed.txt' data/letters.txt
cat changed.txt
```

Actual output

Captured run · exit 0

```
RED
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`s///w FILE`	Write a pattern space only after that substitution succeeds.

Download: SED-14-005.sh

SED-14-006

Edit an owned copy with a backup

The common BSD/GNU -i.bak form changes only the reconstructed input inside the temporary directory. Display the backup too.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-14-005 · Write only successfully changed lines.

Code

```
sed -i.bak 's/red/RED/' data/letters.txt
printf 'edited:\n'
cat data/letters.txt
printf 'backup:\n'
cat data/letters.txt.bak
```

Actual output

Captured run · exit 0

```
edited:
RED
blue
green
backup:
red
blue
green
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`-i.bak`	Keep a backup when editing the owned copy; this option is not POSIX.

Download: SED-14-006.sh

15 - Compose small cleanup recipes

Use a documented tiny text format. Trim or remove one class of text at a time before composing those changes.

SED-15-001

Trim leading whitespace

Change only the beginning of each line; leave trailing spaces intact.

Input

data/padded.txt

```
   red
        blue
 END
```

Code

```
sed 's/^[[:space:]]*//' data/padded.txt
```

Actual output

Captured run · exit 0

```
red
blue
END
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`Leading anchor`	Remove leading whitespace from this line-oriented input.

Download: SED-15-001.sh

SED-15-002

Trim trailing whitespace

Anchor at the other end and preserve the indentation.

Input

data/padded.txt

```
   red
        blue
 END
```

Compare with SED-15-001 · Trim leading whitespace.

Code

```
sed 's/[[:space:]]*$//' data/padded.txt
```

Actual output

Captured run · exit 0

```
  red
       blue
END
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`Trailing anchor`	Remove trailing whitespace, not leading indentation.

Download: SED-15-002.sh

SED-15-003

Compose both trim operations

Use two -e programs in the intended order.

Input

data/padded.txt

```
   red
        blue
 END
```

Compare with SED-15-002 · Trim trailing whitespace.

Code

```
sed -e 's/^[[:space:]]*//' -e 's/[[:space:]]*$//' data/padded.txt
```

Actual output

Captured run · exit 0

```
red
blue
END
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`Two small edits`	Express leading and trailing cleanup as separate operations.

Download: SED-15-003.sh

SED-15-004

Remove complete comment records

The fixture has a tiny unquoted format; only whole comment lines are removed.

Input

data/settings.txt

```
# title
mode=old
note=old
# end
```

Code

```
sed '/^[[:space:]]*#/d' data/settings.txt
```

Actual output

Captured run · exit 0

```
mode=old
note=old
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`Comment address`	Do not treat this as a parser for every configuration syntax.

Download: SED-15-004.sh

SED-15-005

Update one exact setting key

The address keeps the same value on a different key unchanged.

Input

data/settings.txt

```
# title
mode=old
note=old
# end
```

Compare with SED-15-004 · Remove complete comment records.

Code

```
sed '/^mode=/s/=old$/=new/' data/settings.txt
```

Actual output

Captured run · exit 0

```
# title
mode=new
note=old
# end
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`Key plus value shape`	Constrain both the record selected and the part replaced.

Download: SED-15-005.sh

SED-15-006

Collapse horizontal space runs

An ERE repetition maps every run of two or more spaces to one.

Input

data/spaces.txt

```
a  b   c
keep
```

Compare with SED-13-006 · Repeat a substitution while it makes progress.

Code

```
sed -E 's/ {2,}/ /g' data/spaces.txt
```

Actual output

Captured run · exit 0

```
a b c
keep
```

Parameter	Meaning in this example
`Implementation`	BSD/GNU common
`Run normalization`	Use a direct repetition rule instead of a substitution loop.

Download: SED-15-006.sh

16 - GNU sed extensions

These examples explicitly select GNU sed. They keep extension syntax apart from the common BSD/GNU instructions taught earlier.

SED-16-001

Let a regex range end on the first line

GNU address zero permits the ending regex to match line one immediately.

Input

data/starts.txt

```
START
middle
START
end
```

Compare with SED-03-005 · Inspect a range whose first line matches its end regex.

Code

```
gsed -n '0,/^START$/p' data/starts.txt
```

Actual output

Captured run · exit 0

```
START
```

Parameter	Meaning in this example
`Implementation`	GNU sed
`0,/regex/`	Begin before line one so the first line can close this GNU range.

Download: SED-16-001.sh

SED-16-002

Select lines by a step address

GNU step addresses can select every other line without a hold-space program.

Input

data/lines.txt

```
one
two
three
four
five
six
```

Code

```
gsed -n '1~2p' data/lines.txt
```

Actual output

Captured run · exit 0

```
one
three
five
```

Parameter	Meaning in this example
`Implementation`	GNU sed
`1~2`	Start at line one and select every second line.

Download: SED-16-002.sh

SED-16-003

Convert replacement text to uppercase

The GNU replacement escape changes the matched text to uppercase.

Input

data/letters.txt

```
red
blue
green
```

Code

```
gsed 's/.*/\U&/' data/letters.txt
```

Actual output

Captured run · exit 0

```
RED
BLUE
GREEN
```

Parameter	Meaning in this example
`Implementation`	GNU sed
`\U`	Apply GNU uppercase conversion to the following replacement text.

Download: SED-16-003.sh

SED-16-004

Quit without automatically printing the current line

GNU Q stops immediately without the current automatic output. Earlier lines already printed remain.

Input

data/letters.txt

```
red
blue
green
```

Compare with SED-09-005 · Quit after a selected line.

Code

```
gsed '2Q' data/letters.txt
```

Actual output

Captured run · exit 0

```
red
```

Parameter	Meaning in this example
`Implementation`	GNU sed
`Q`	Contrast quiet GNU quit with the common q instruction.

Download: SED-16-004.sh

SED-16-005

Restart line numbering for separate files

GNU -s processes these files as separate streams, so address one selects the first line of each.

Input

data/first.txt

```
one
two
```

data/second.txt

```
three
four
```

Code

```
gsed -sn '1p' data/first.txt data/second.txt
```

Actual output

Captured run · exit 0

```
one
three
```

Parameter	Meaning in this example
`Implementation`	GNU sed
`-s`	Use separate-file address and numbering scope.

Download: SED-16-005.sh

SED-16-006

Treat NUL as the record separator

GNU -z sees two records. The final tr only turns their terminators into visible output lines.

Code

```
printf 'alpha\000beta\000' | gsed -z 's/^/item=/' | tr '\000' '\n'
```

Actual output

Captured run · exit 0

```
item=alpha
item=beta
```

Parameter	Meaning in this example
`Implementation`	GNU sed
`-z`	Apply the substitution once to each NUL-delimited record.

Download: SED-16-006.sh

Example and command index

- SED-01-001 · Copy input with an empty program
- SED-01-002 · Suppress automatic printing
- SED-01-003 · Print explicitly while automatic output is active
- SED-01-004 · Use one explicit output path
- SED-01-005 · Pass a program with -e
- SED-01-006 · Read the same program from a file
- SED-02-001 · Print the first input line
- SED-02-002 · Print the second input line
- SED-02-003 · Print the final input line
- SED-02-004 · Print an inclusive numeric range
- SED-02-005 · Continue the range through the end
- SED-02-006 · Invert a single address
- SED-03-001 · Print lines matching an address
- SED-03-002 · Anchor the address at the start
- SED-03-003 · Print a delimited inclusive region
- SED-03-004 · Continue from a marker to the input end
- SED-03-005 · Inspect a range whose first line matches its end regex
- SED-03-006 · Print the complement of an addressed set
- SED-04-001 · Replace the first occurrence on a line
- SED-04-002 · Replace every nonoverlapping occurrence
- SED-04-003 · Replace only the second occurrence
- SED-04-004 · Replace a match with an empty string
- SED-04-005 · Choose a delimiter that fits the data
- SED-04-006 · Write the same change with slash delimiters
- SED-05-001 · Observe a wildcard period
- SED-05-002 · Require a literal period
- SED-05-003 · Choose characters with a class
- SED-05-004 · Repeat a BRE atom
- SED-05-005 · Use a BRE interval
- SED-05-006 · Switch to ERE for the same interval
- SED-06-001 · Insert the whole match into replacement text
- SED-06-002 · Insert a literal ampersand
- SED-06-003 · Capture a name using BRE groups
- SED-06-004 · Use ERE groups for the same capture
- SED-06-005 · Reorder two captured fields
- SED-06-006 · Reuse the same capture twice
- SED-07-001 · Print only successful substitutions
- SED-07-002 · See p and automatic printing together
- SED-07-003 · Combine global substitution with conditional printing
- SED-07-004 · Print only if a second occurrence exists
- SED-07-005 · Restrict substitution with a regex address
- SED-07-006 · Restrict substitution with a numeric address
- SED-08-001 · Delete one addressed line
- SED-08-002 · Delete an inclusive range
- SED-08-003 · Delete only empty lines
- SED-08-004 · Delete whitespace-only lines too
- SED-08-005 · Insert text before a selected line
- SED-08-006 · Append text after a selected line
- SED-09-001 · Replace one complete line
- SED-09-002 · Replace a whole addressed range once
- SED-09-003 · Translate one character everywhere
- SED-09-004 · Print input line numbers
- SED-09-005 · Quit after a selected line
- SED-09-006 · Inspect pattern space with visible line endings

- SED-10-001 · Give one command a regex scope
- SED-10-002 · Add a second instruction to that scope
- SED-10-003 · Negate the scope of a group
- SED-10-004 · Share a numeric range across a group
- SED-10-005 · Print and then end the selected cycle
- SED-10-006 · Observe that deletion skips later instructions
- SED-11-001 · Save a line and retrieve it later
- SED-11-002 · Retrieve without changing the saved buffer
- SED-11-003 · Exchange pattern space and hold space
- SED-11-004 · Append a line to hold space
- SED-11-005 · Append hold space to the current pattern
- SED-11-006 · Retrieve the initially empty hold space
- SED-12-001 · Advance with lowercase n
- SED-12-002 · Print the line read by n explicitly
- SED-12-003 · Append the next line with uppercase N
- SED-12-004 · Print only the first part with P
- SED-12-005 · Restart after dropping the first part with D
- SED-12-006 · Join each complete pair with a space
- SED-13-001 · Branch to the end of a program
- SED-13-002 · Branch only on a selected record
- SED-13-003 · Jump to a named label
- SED-13-004 · Branch after a successful substitution
- SED-13-005 · Skip a fallback edit after success
- SED-13-006 · Repeat a substitution while it makes progress
- SED-14-001 · Preview an edit before writing a file
- SED-14-002 · Save transformed output to another file
- SED-14-003 · Read extra text after an addressed line
- SED-14-004 · Write selected pattern spaces to a file
- SED-14-005 · Write only successfully changed lines
- SED-14-006 · Edit an owned copy with a backup
- SED-15-001 · Trim leading whitespace
- SED-15-002 · Trim trailing whitespace
- SED-15-003 · Compose both trim operations
- SED-15-004 · Remove complete comment records
- SED-15-005 · Update one exact setting key
- SED-15-006 · Collapse horizontal space runs
- SED-16-001 · Let a regex range end on the first line
- SED-16-002 · Select lines by a step address
- SED-16-003 · Convert replacement text to uppercase
- SED-16-004 · Quit without automatically printing the current line
- SED-16-005 · Restart line numbering for separate files
- SED-16-006 · Treat NUL as the record separator

Content: CC0 1.0. Structure and publishing framework: CC BY 4.0 by Lukas C. Bossert. Repository source · Publishing framework source · License scope and third-party notices.