Linux ed and red commands

Updated: 11/06/2021 by Computer Hope

On Unix-like operating systems, **ed** is an interactive file editor.

red, short for "restricted ed", is a version of **ed** which carries restrictions about what types of editing can be performed.

Description

ed is one of the oldest editing programs around. It was introduced in 1969, approximately 50 years ago, as one of the original components of Unix.

Syntax

```
ed [options] [file]
```

```
red [options] [file]
```

Options

-h, --help	Display a help message and exit.
-V, --version	Output version information and exit.
-G, --traditional	Run in compatibility mode.
-l, --loose-exit-status	Exit with a status of zero (normal termination) even if a command fails. This option can be useful if **ed** is set to be the editor for **crontab**, for instance.
-p, --prompt=*STRING*	**ed** usually waits for user input at a blank line; this option uses the string STRING as the prompt instead.
-r, --restricted	Run in restricted mode.
-s, --quiet, --silent	Suppress diagnostics.
-v, --verbose	Operate verbosely.

Technical description

ed is a line-oriented text editor with a minimal interface. It is used to create, display, modify and otherwise manipulate text files.

If **ed** is invoked with a file name argument, then a copy of file is read into the editor's buffer. Changes are made to this copy and not directly to file itself. Upon quitting **ed**, any changes not explicitly saved with a '**w**' command are lost.

Editing is performed in two distinct modes: **command** and **input**. When first invoked, **ed** is in **command mode**. In this mode, commands are read from the standard input and executed to manipulate the contents of the editor buffer. A typical command might look like:

```
,s/old/new/g
```

Which replaces all occurrences of the string *old* with the string *new*.

When an input command, such as '**a**' (append), '**i**' (insert), or '**c**' (change) is given, **ed** enters **input mode**. This mode is the primary means of adding text to a file. In this mode, no commands are available; instead, the standard input is written directly to the editor buffer. Lines consist of text up to and including a newline character. Input mode is exited by entering a single period ("**.**") on a line.

An important thing to remember is that all **ed** commands operate on whole lines or ranges of lines. For example, the '**d**' command deletes lines; the '**m**' command moves lines, etc. It is possible to modify only a portion of a line by specific replacement, as in the example above; although even then, the '**s**' command is applied to whole lines at a time.

In general, **ed** commands consist of zero or more line addresses, followed by a single character command and possibly additional parameters. Commands have the structure:

```
[address [,address]]command[parameters]
```

The address(es) indicate the line or range of lines to be affected by the command. If fewer addresses are given than the command accepts, then default addresses are supplied.

Line addresses

An *address* represents the number of a line in the buffer. **ed** maintains a current address that is supplied to commands as the default address if none other is specified.

When a file is first read, the current address is set to the last line of the file. In general, at any given time, the current address is set to the last line affected by a command.

Think of the address as a place marker to the last line in the file where **ed** has done something, and, unless told otherwise, the line where it operates next.

Referring to line addresses

A reference to a line address is constructed from one of the bases in the list below, optionally followed by a numeric offset. The offset may include any combination of digits, operators (for instance, "**+**" and "**-**") and whitespace. Addresses are read from left to right, and their values are computed relative to the current address.

One exception to the rule that addresses represent line numbers is the special address **0** (zero). This address means "before the first line," and can be used in any command.

Address ranges

An *address range* is two addresses separated either by a comma ("**,**") or semicolon ("**;**"). The value of the first address in a range cannot exceed the value of the second. If only one address is given in a range, then the second address is set to the given address.

Each address in a comma-delimited range is interpreted relative to the current address. In a semicolon-delimited range, the first address is used to set the current address, and the second address is interpreted relative to the first.

The following address symbols are recognized:

.	The current line (address) in the buffer.
$	The last line in the buffer.
n	The *n*th line in the buffer where *n* is a number in the range **[0,$]**.
-	The previous line. This option is equivalent to **-1** and may be repeated with cumulative effect.
^*n*	The *n*th previous line, where *n* is a non-negative number.
+	The next line. This option is equivalent to **+1** and may be repeated with cumulative effect.
+*n*	The *n*th next line, where *n* is a non-negative number.
,	The first-through-last lines in the buffer. This option is equivalent to the address range **1,$**.
;	The current through last lines in the buffer. This option is equivalent to the address range **.,$**.
/*re***/**	The next line containing the regular expression *re*. The search wraps to the beginning of the buffer and continues down to the current line, if necessary. **//** repeats the last search.
?*re***?**	The previous line containing the regular expression *re*. The search wraps to the end of the buffer and continues up to the current line, if necessary. **??** repeats the last search.
'*lc*	The line previously marked by a '**k**' (mark) command, where *lc* is a lowercase letter.

Regular expressions

Regular expressions are patterns used in selecting text. For example, the **ed** command

```
g/string/
```

Prints all lines containing **string**. Regular expressions are also used by the '**s**' command for selecting old text to be replaced with new. In addition to a specifying string literals, regular expressions can represent classes of strings. If it is possible for a regular expression to match several strings in a line, then the left-most longest match is the one selected.

The following symbols are used in constructing regular expressions:

c	Any character *c* not listed below, including '**{**', '**}**', '**(**', '**)**', '**<**' and '**>**', matches itself.
*****c*	A backslash-escaped character *c* other than '**{**', '**}**', '**(**', '**)**', '**<**', '**>**', '**b**', '**B**', '**w**', '**W**', '**+**', and '**?**' matches itself.
.	Matches any single character.

[*char-class***]**	Matches any single character in *char-class*. To include a '**]**' in *char-class*, it must be the first character. A range of characters may be specified by separating the end characters of the range with a '**-**'; for instance, '**a-z**' specifies the lowercase characters. The following literal expressions can also be used in *char-class* to specify sets of characters:
	
	`[:alnum:][:cntrl:][:lower:][:space:][:alpha:][:digit:]`
	`[:print:][:upper:][:blank:][:graph:][:punct:][:xdigit:]`
	
	If '**-**' appears as the first or last character of *char-class*, then it matches itself. All other characters in *char-class* match themselves.
	
	Patterns in *char-class* of the form:
	
	`[.col-elm.]`
	
	or,
	
	`[=col-elm=]`
	
	where *col-elm* is a collating element interpreted according to the locale of the system.

[^*char-class***]**	Matches any single character, other than newline, not in *char-class*. *char-class* is defined as above.
^	If '**^**' is the first character of a regular expression, then it anchors the regular expression to the beginning of a line. Otherwise, it matches itself.

$	If '**$**' is the last character of a regular expression, it anchors the regular expression to the end of a line. Otherwise, it matches itself.
\\(*re***\\)**	Defines a (possibly null) subexpression *re*. Subexpressions may be nested. A subsequent backreference of the form '*****n*', where *n* is a number in the range **[1,9]**, expands to the text matched by the *n*th subexpression. For example, the regular expression '**\\(a.c\\)\\1**' matches the string '**abcabc**', but not '**abcadc**'. Subexpressions are ordered relative to their left delimiter.
*****	Matches the single-character regular expression or subexpression immediately preceding it zero or more times. If '*****' is the first character of a regular expression or subexpression, then it matches itself. The '*****' operator sometimes yields unexpected results. For example, the regular expression '**b***' matches the beginning of the string '**abbb**', instead of the substring '**bbb**', since a null match is the only left-most match.
\\{*n***,***m***\\}** **\\{***n***,\\}** **\\{***n***\\}**	Matches the single character regular expression or subexpression immediately preceding it at least *n* and at most *m* times. If *m* is omitted, then it matches at least *n* times. If the comma is also omitted, then it matches exactly *n* times. If any of these forms occurs first in a regular expression or subexpression, then it is interpreted literally (i.e., the regular expression '**\\{2\\}**' matches the string '**{2}**', and so on).
\\< **\\>**	Anchors the single character regular expression or subexpression immediately following it to the beginning (**\\<**) or ending (**\\<**) of a word, i.e., in ASCII, a maximal string of alphanumeric characters, including the underscore ("**_**").

The following extended operators are preceded by a backslash (****) to distinguish them from traditional **ed** syntax.

\\' **\\'**	Unconditionally matches the beginning (**\\'**) or ending (**\\'**) of a line.
\\?	Optionally matches the single character regular expression or subexpression immediately preceding it. For example, the regular expression '**a[bd]\\?c**' matches the strings '**abc**', '**adc**' and '**ac**'. If **\\?** occurs at the beginning of a regular expressions or subexpression, then it matches a literal '**?**'.
\\+	Matches the single character regular expression or subexpression immediately preceding it one or more times. So the regular expression '**a\\+**' is shorthand for '**aa***'. If **\\+** occurs at the beginning of a regular expression or subexpression, then it matches a literal '**+**'.
\\b	Matches the beginning or ending (null string) of a word. Thus the regular expression '**\\bhello\\b**' is equivalent to '**\\<hello\\>**'. However, '**\\b\\b**' is a valid regular expression whereas '**\\<\\>**' is not.
\\B	Matches (a null string) inside a word.
\\w	Matches any character in a word.

\W	Matches any character not in a word.

Commands

All **ed** commands are single characters, though some require additional parameters. If a command's parameters extend over several lines, then each line except for the last must be terminated with a backslash (****).

In general, at most one command is allowed per line. However, most commands accept a print suffix, which is any of '**p**' (print), '**l**' (list), or '**n**' (enumerate), to print the last line affected by the command.

An interrupt (typically **^C**) has the effect of aborting the current command and returning the editor to command mode.

ed recognizes the following commands. The commands are shown together with the default address or address range supplied if none is specified (in brackets).

[**.**,**.**]**a**	Appends text to the buffer after the addressed line, which may be the address **0** (zero). Text is entered in input mode. The current address is set to last line entered.
[**.**,**.**]**c**	Changes lines in the buffer. The addressed lines are deleted from the buffer, and text is appended in their place. Text is entered in input mode. The current address is set to last line entered.
[**.**,**.**]**d**	Deletes the addressed lines from the buffer. If there is a line after the deleted range, then the current address is set to this line. Otherwise, the current address is set to the line before the deleted range.
[**.**,**.**]**e** *file*	Edits *file*, and sets the default file name. If *file* is not specified, then the default file name is used. Any lines in the buffer are deleted before the new file is read. The current address is set to the last line read.
e !*command*	Edits the standard output of '**!***command*', (see **!***command* below). The default file name is unchanged. Any lines in the buffer are deleted before the output of command is read. The current address is set to the last line read.
E *file*	Edits *file* unconditionally. This option is similar to the **e** command, except that unwritten changes are discarded without warning. The current address is set to the last line read.
f *file*	Sets the default file name to *file*. If *file* is not specified, then the default unescaped file name is printed.
[**1**,**$**]**g/***re***/***command-list*	Applies *command-list* to each of the addressed lines matching a regular expression *re*. The current address is set to the line currently matched

	before *command-list* is executed. At the end of the '**g**' command, the current address is set to the last line affected by *command-list*. Each command in *command-list* must be on a separate line, and every line except for the last must be terminated by a backslash (****). Any commands are allowed, except for '**g**', '**G**', '**v**', and '**V**'. A newline alone in *command-list* is equivalent to a '**p**' command.
[**1,$**]**G/***re***/**	Interactively edits the addressed lines matching a regular expression *re*. For each matching line, the line is printed, the current address is set, and the user is prompted to enter a *command-list*. At the end of the '**G**' command, the current address is set to the last line affected by (the last) *command-list*. The format of *command-list* is the same as that of the '**g**' command. A newline alone acts as a null command list. A single '**&**' repeats the last non-null command list.
H	Toggles the printing of error explanations. By default, explanations are not printed. It is recommended that **ed** scripts begin with this command to aid in debugging.
h	Prints an explanation of the last error.
[**.**]**i**	Inserts text in the buffer before the current line. Text is entered in input mode. The current address is set to the last line entered.
[**.,.+1**]**j**	Joins the addressed lines. The addressed lines are deleted from the buffer and replaced by a single line containing their joined text. The current address is set to the resultant line.
[**.**]**k***c*	Marks a line with a lowercase letter *c*. The line can then be addressed as **'***c* (a single quote followed by *c*) in subsequent commands. The mark is not cleared until the line is deleted or otherwise modified.
[**.,.**]**l**	Prints the addressed lines unambiguously. If invoked from a terminal, **ed** pauses at the end of each page until a newline is entered. The current address is set to the last line printed.
[**.,.**]**m**[**.**]	Moves lines in the buffer. The addressed lines are moved to after the right destination address, which may be the address **0** (zero). The current address is set to the last line moved.
[**.,.**]**n**	Prints the addressed lines with their line numbers. The current address is set to the last line printed.
[**.,.**]**p**	Prints the addressed lines. If invoked from a terminal, **ed** pauses at the end of each page until a newline is entered. The current address is set to the last line printed.
P	Toggles the command prompt on and off. Unless a prompt was specified by with command-line option **-p** *string*, the command prompt is by default turned off.

q	Quits **ed**.
Q	Quits **ed** unconditionally. This option is similar to the **q** command, except that unwritten changes are discarded without warning.
[**$**]**r** *file*	Reads *file* to after the addressed line. If *file* is not specified, then the default file name is used. If there was no default file name before the command, then the default file name is set to *file*. Otherwise, the default file name is unchanged. The current address is set to the last line read.
[**$**]**r** *!command*	Reads to after the addressed line the standard output of '*!command*', (see the *!command* below). The default file name is unchanged. The current address is set to the last line read.
[**.**,**.**]**s/***re***/***replacement***/** [**.**,**.**]**s/***re***/***replacement***/g** [**.**,**.**]**s/***re***/***replacement***/***n*	Replaces text in the addressed lines matching a regular expression *re* with replacement. By default, only the first match in each line is replaced. If the '**g**' (global) suffix is given, then every match to be replaced. The '*n*' suffix, where *n* is a positive number, causes only the *n*th match to be replaced. It is an error if no substitutions are performed on any of the addressed lines. The current address is set the last line affected. The *re* and *replacement* may be delimited by any character other than space and newline (see the '**s**' command below). If one or two of the last delimiters is omitted, then the last line affected is printed as though the print suffix '**p**' were specified. An unescaped '**&**' in replacement is replaced by the currently matched text. The character sequence '*****m*', where *m* is a number in the range **[1,9]**, is replaced by the *m*th backreference expression of the matched text. If replacement consists of a single '**%**', then replacement from the last substitution is used. Newlines may be embedded in replacement if they are escaped with a backslash (****).
[**.**,**.**]**s**	Repeats the last substitution. This form of the '**s**' command accepts a count suffix '**n**', or any combination of the characters '**r**', '**g**', and '**p**'. If a count suffix '**n**' is given, then only the *n*th match is replaced. The '**r**' suffix causes the regular expression of the last search to be used instead of the that of the last substitution. The '**g**' suffix toggles the global suffix of the last substitution. The '**p**' suffix toggles the print suffix of the last substitution. The current address is set to the last line affected.
[**.**,**.**]**t**[**.**]	Copies the addressed lines to after the right destination address, which may be the address **0** (zero). The current address is set to the last line copied.
u	Undoes the last command and restores the current address to what it was before the command. The global commands '**g**', '**G**', '**v**', and '**V**' are treated as a single command by undo. '**u**' is its own inverse.

[1,$]v/*re***/***command-list*	Applies *command-list* to each of the addressed lines not matching a regular expression *re*. This option is similar to the '**g**' command.
[1,$]V/*re***/**	Interactively edits the addressed lines not matching a regular expression *re*. This option is similar to the '**G**' command.
[1,$]w *file*	Writes the addressed lines to file. Any previous contents of file is lost without warning. If there is no default file name, then the default file name is set to *file*, otherwise it is unchanged. If no file name is specified, then the default file name is used. The current address is unchanged.
[1,$]wq *file*	Writes the addressed lines to file, and then executes a '**q**' command.
[1,$]w *!command*	Writes the addressed lines to the standard input of '*!command*', (see the *!command* below). The default file name and current address are unchanged.
[1,$]W *file*	Appends the addressed lines to the end of file. This option is similar to the '**w**' command, expect that the previous contents of file is not clobbered. The current address is unchanged.
[**.**]x	Copies (puts) the contents of the cut buffer to after the addressed line. The current address is set to the last line copied.
[**.**,**.**]**y**	Copies (yanks) the addressed lines to the cut buffer. The cut buffer is overwritten by subsequent '**y**', '**s**', '**j**', '**d**', or '**c**' commands. The current address is unchanged.
[**.+1**]z*n*	Scrolls *n* lines at a time starting at addressed line. If *n* is not specified, then the current window size is used. The current address is set to the last line printed.
!*command*	Executes *command* via **sh**. If the first character of *command* is '**!**', then it is replaced by text of the previous '*!command*'. **ed** does not process *command* for backslash (****) escapes. However, an unescaped '**%**' is replaced by the default file name. When the shell returns from execution, a '**!**' is printed to the standard output. The current line is unchanged.
[**.**,**.**]**#**	Begins a comment; the rest of the line, up to a newline, is ignored. If a line address followed by a semicolon is given, then the current address is set to that address. Otherwise, the current address is unchanged.
[**$**]**=**	Prints the line number of the addressed line.
[**.+1**]	Prints the addressed line, and sets the current address to that line.

Restricted ed

red operates like **ed**, except that it can only edit files in the current directory, and it cannot execute shell commands.

Examples

Getting started

The simplest way to start **ed** is to run it with no options at the command line:

```
ed
```

This command places your cursor at a new line, but you won't receive any other prompt or indication you're in **ed**. To quit **ed** and return to your shell's command prompt, type **q** and press **Enter**:

```
q
```

You could also have quit using a capital **Q**, which is an "unconditional" quit: it exits regardless if you have changes that haven't yet been written to a file.

Let's go back into **ed**:

```
ed
```

Now let's intentionally type a command that **ed** won't recognize, to cause an error. There is no **o** command, so let's type that:

```
o
```

```
?
```

The "**?**" means that something went wrong. By default, **ed** has such a minimal interface that it doesn't give you a specific error message. If an error has occurred, you can find out what happened using the **h** command:

```
h
```

```
Unknown command
```

And then **ed** is ready for another command.

To turn on error messages so they always display immediately by default, use the **H** command:

```
H
```

Making ed more friendly

It's easier to use **ed** if you have a command prompt. Let's quit **ed**...

```
q
```

...which returns us to our shell's command prompt. Now let's start **ed** again using the **-p** option, which tells **ed** to use a command prompt. We specify what prompt we want to use as a string immediately following **-p**. For instance, let's use the simple prompt '**>** '. Here's how we invoke **ed**:

```
ed -p'> '
```

Now when **ed** launches, we see our prompt:

```
>
```

Adding text, and viewing it

We haven't given **ed** a file name, so it starts us with an empty buffer to enter text. Let's enter some now. We use the **a** command to add text; in the commands below we also show the prompt before the command, for clarity.

```
> a
```

Now we are given blank lines where we can type our text. We enter it like in any word processor, pressing **Enter** at the end of each line. When we're done, we enter a period alone on a new line, to tell **ed** we're done:

```
Line one,
line two,
line three,
line four.
.
```

We've entered four lines; the editor leaves off there, and gives us another **ed** command prompt. If we enter the **p** command, it prints the line where we left off editing:

```
> p
```

```
line four.
```

Most **ed** commands can be prefixed with an address range to specify what range to apply to the text. So, if we want to print lines **1** through **4**, we could use the command **1,4p**:

```
> 1,4p
```

```
Line one,
line two,
line three,
line four.
```

If we omit the line numbers (but leave in the comma), **ed** assumes maximum start and end values, so it prints the whole buffer:

```
> ,p
```

```
Line one,
line two,
line three,
line four.
```

In case we forget what line number we left off at, we can use the **=** command:

```
> =
```

```
4
```

We can continue to add lines using the **a** command again, and entering a period on a new line when we're done:

```
> a
```

```
Line five.
.
```

```
> ,p
```

```
Line one,
line two,
line three,
line four.
Line five.
```

We're now at line 5:

```
> =
```

```
5
```

We can change to another line by entering that line number. When we do, **ed** prints that line to let us know where we are:

```
> 3
```

```
line three,
```

Now if we entered **a**, we'd start adding lines after line 3. Or we can use the **i** command to start inserting lines *before* line 3:

```
> i
```

```
This is the new line three. All lines after this are off by one!
.
```

Let's check to see where the editor leaves us after this insertion. This is known as where the buffer is "addressed," and it should be the last line of the buffer:

```
> =
```

```
6
```

Let's look at all six lines:

```
> ,p
```

```
Line one,
line two,
This is the new line three. All lines after this are off by one!
```

```
line three,
line four.
Line five.
```

Let's delete the third line using the **d** command. We put a **3** before the **d**, to specify that line 3 is the line to be deleted:

```
> 3d
```

...and view the changed buffer:

```
,p
```

```
Line one,
line two,
line three,
line four.
Line five.
```

To view lines with the line number at the beginning, use the **n** command. For example:

```
> n
```

```
5       Line five.
```

```
> 3n
```

```
3       line three,
```

```
> 2,4n
```

```
2     line two,
3     line three,
4     line four.
```

To change a line (completely replace it), you can use the **c** command. You can enter multiple lines of text, so you still need to put a period on a new line to indicate you're done:

```
> 3c
```

```
line the third,
.
```

It also works on multiple lines:

```
> 1,2c
```

```
First line,
second line,
.
```

```
> 4,5c
```

```
line
 four.
.
```

And we can join lines with the **j** command:

```
> 4,5j
```

Let's look at those changes:

```
> ,p
```

```
First line,
second line,
line the third,
line four.
```

Understanding ranges

Let's explore ranges a little more. Let's select line 2, then print one line before and after it:

```
> 2
```

```
second line,
> -1,+1p
```

```
First line,
second line,
line the third,
```

We've already seen that we can specify the entire buffer using a single comma as the range ("**,**"):

```
> ,p
```

```
First line,
second line,
line the third,
line four.
```

We've been using a single comma ("**,**") to represent the entire buffer range in our commands. A percent sign ("**%**") does the same thing:

```
> %p
```

```
First line,
second line,
line the third,
line four.
```

When specifying ranges, we can represent the last line using a dollar sign ("**$**"):

```
> 2,$p
```

```
second line,
line the third,
line four.
```

If you don't specify a range, most commands assume a range of the currently-addressed line. You can also specify the currently-addressed line explicitly using a period ("**.**"):

```
> 2
```

```
second line,
> .
```

```
second line,
```

```
> .=
```

```
2
```

```
> .p
```

```
second line,
```

Marking lines for easier reference

You can mark a line using the **k** command, giving it a "marker" of any lowercase letter. You can then refer to the marked line using a single quote followed by that letter. For instance, here we mark line **3** with the letter **a**, then print that line by referring to it with **'a**:

```
> 3ka
```

```
> 'ap
```

```
line the third,
```

Moving and copying lines

You can move lines from one place to another using the **m** command:

```
> 1,2m3
```

```
> ,p
```

```
line the third,
First line,
second line,
line four.
```

You can copy lines to another location using the **t** command:

```
> 2t4
```

```
> ,p
```

```
line the third,
First line,
second line,
line four.
First line,
```

Writing to and reading from files

When you're ready to write to a file, you can use the **f** command to specify a file name:

```
> f myfile.txt
```

```
myfile.txt
```

We can now write to **myfile.txt** using the **w** command. **ed** tells us how many bytes were written:

```
> w
```

```
64
```

If we specify a file name after **w**, **ed** writes to that file name, and tell us how many bytes:

```
> w myfile2.txt
```

```
64
```

To open a file for editing, use the **e** command with the name of the file you want to open. **ed** reads the file into the buffer, and let you know how many bytes were read in:

```
> e myfile.txt
```

```
64
```

If you try to open a file for editing when you have unsaved changes in your current buffer, you receive an error. To open a file with unsaved changes, losing any unwritten changes to the current buffer, use a capital **E**:

```
> E myfile.txt
```

64

The same goes for exiting **ed**: if you try to use the **q** command to exit and you have unsaved changes, you receive an error. To quit without writing your changes, use a capital **Q**:

```
> Q
```

To open a file from the command line, specify it with the last argument when invoking **ed**. The file is read into the buffer, and **ed** reports how many bytes were read and give you a prompt for a command. For example:

```
ed -p'> ' myfile.txt
```

```
64
>
```

Searching for and replacing text

Let's delete all the lines in our buffer and re-make the file before the next examples:

```
> ,d
```

```
> a
```

```
Twinkle, twinkle, little star.
How I wonder what you are.
Although I suspect you're mostly made of hydrogen.
.
```

To search for text in your file, use the **/** command with the text to search for:

```
> /twinkle
```

```
Twinkle, twinkle, little star.
```

To globally run a command on lines matching a regular expression, use the command form **g/***regular-expression***/***command*. For instance, here we'll **g**lobally search for lines containing the consecutive letters "**re**", and **p**rint those lines:

```
> g/re/p
```

```
How I wonder what you are.
Although I suspect you're mostly made of hydrogen.
```

To run a command on the lines which do *not* match a specific regular expression, use the command form **v/***regular-expression***/***command*:

```
> v/re/p
```

```
Twinkle, twinkle, little star.
```

To perform text substitution, use the command form **s/***text-to-replace***/***replacement-text***/**. We specify a range before the **s** so that it searches more than only the currently-addressed line:

```
> ,s/hydrogen/hydrogen and helium/
```

```
> ,p
```

```
Twinkle, twinkle, little star.
How I wonder what you are.
Although I suspect you're mostly made of hydrogen and helium.
```

By default, only the first occurrence of *text-to-replace* in a line is substituted. To replace every occurrence in each matching line, specify **g** after the second slash, for a "global" substitution:

```
> ,s/wink/wonk/g
```

```
> 1p
```

```
Twonkle, twonkle, little star.
```

Conclusion

Hopefully this has given you enough familiarity with **ed** to be able to use it, and to make use of the references available above. **ed** can be very useful when working on a system with minimal system resources, or when repairing a system that cannot boot beyond a very basic runlevel. It's the progenitor of many modern editors, and once you get used to it, its simplicity can make basic, important tasks easy to perform.

Related commands

bfs — An editor that can load very large files.
cat — Output the contents of a file.
edit — A text editor.
ex — Line-editor mode of the **vi** text editor.
grep — Filter text which matches a regular expression.
ksh — The Korn shell command interpreter.
pico — A simple text editor.
sed — A utility for filtering and transforming text.
sh — The Bourne shell command interpreter.
stty — Set options for your terminal display.
umask — Get or set the file mode creation mask.
vi — Text editor based on the visual mode of **ex**.
vim — An advanced version of **vi**.