

vi(1) Tips

Essential vi/vim Editor Skills

by

Jacek Artymiak

vi(1) Tips

Essential vi/vim Editor Skills

by

Jacek Artymiak

First Edition

Lublin 2008

vi(1) Tips, 1st ed. by Jacek Artymiak

Editor: Suzanne Trellis

Published by devGuide.net

www: <http://www.devguide.net/books/vitips1>

email: jacek@devguide.net

Copyright © 2008 Jacek Artymiak

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

First edition 2008

Printed in Poland and the United States of America.

08 10 9 8 7 6 5 4 3 2 1

ISBN: 978-83-60869-00-0

The author and the publisher disclaim any and all liability for the use of information and programs contained in this book.

All trademarks mentioned in this book are the sole property of their owners.

Table of Contents

Introduction	I
Chapter 1: Essentials	3
PANIC!	5
Canceling Commands	5
Switching Between Command Mode and Insert Mode	6
Escaping from the ex Editor Mode	6
Unscrambling the Screen	6
Chapter 2: Basic File Operations	7
Starting vi(1)	9
Starting vi(1) and Opening a File for Editing	10
Starting vi(1) and Opening Multiple Files for Editing	11
Switching Between Files	13
Opening a File after Starting vi(1)	14
Saving the Current File	15
Forcing vi(1) to Save the Current File	16
Saving the Current File Under a Different Name	17
Saving a Part of the Current File	18
Saving a Part of the Current File Under a Different Name	19
Appending the Current File to Another	20
Appending a Part of the Current File to Another	21
Saving the Current file and Quitting vi(1)	22
Forcing vi(1) to Save the Current File and Quit	23

Quitting vi(1) without Saving the Current File	24
Forcing vi(1) to Quit without Saving the Current File	25
Recovering the Current File	26
Chapter 3: Cursor Movement	27
Moving the Cursor One Character/Line at a Time	29
Moving the Cursor x Characters or Lines at a Time	30
Moving the Cursor to Column x	32
Moving the Cursor to the Start or End of Line	33
Moving the Cursor Between Lines	34
Which Line Am I On?	36
Moving the Cursor Relative to the vi(1) Screen	38
Moving the Cursor to Character x	40
Moving the Cursor Between Words	41
Moving the Cursor Between Sentences	44
Moving the Cursor Between Paragraphs	45
Moving the Cursor Between Matching 0, {, [, or <>	47
Moving the Cursor Between Markers	48
Moving Around with Simple Search	50
Repositioning Text Relatively to the Screen	51
Chapter 4: Editing	53
Entering Text	55
Inserting Lines	57
Inserting Files	59

Inserting Output of a Command	60
Processing Text Using External Commands	61
Changing Text	63
Replacing Text	67
Replacing One or More Characters with Any Number of Characters	69
Replacing the Current Line	70
Deleting Text	71
Search and Replace	77
Cut, Copy, and Paste	80
Copying Text	80
Pasting Text	84
Joining Lines	85
Changing Case	86
Incrementing and Decrementing Numbers	87
Repeating Actions	87
Undo / Redo	87
Chapter 5: Tricks	89
Running Commands	91
Sending vi(i) to the Background	92
Shell Access	92
Index	95

List of Tables

Table 1: UNIX filename wildcards.	12
Table 2: Motion commands.	66
Table 3: Regular expressions.	76

Introduction

No Unix-class system administrator or user will get far without learning the basics of vi(1), the widespread visual text-mode editor.

Contrary to some misinformed opinions spread among users who are new to Unix-class systems, vi(1) is not difficult to learn. Granted, it is over thirty years old and not very friendly to beginners, but once you grasp the basic concepts, you will never have to learn another text editor again, because vi(1) is available for all standard operating systems, including Microsoft Windows, Mac OS X, Linux, BSD, and many others.

My own experience teaching vi(1) and Unix-class system and network administration suggests that most of the problems reported by new users stem from the fact that vi(1) is completely different from any other text editor.

Vi(1) is a *modal* editor, which means that you need to tell it to switch between different modes and commands, even when all you want to do is change a few characters. This minor inconvenience is offset by the enormous flexibility of the editor and its seamless integration with the rest of the Unix environment.

Another problem is the obscure terminology used to describe its functionality. It makes learning vi(1) unnecessarily difficult for no good reason.

Bearing those two issues in mind, I have written a book which does not bombard you with the old terminology, but rather uses concepts that are familiar to users who are used to working with Microsoft Windows or Mac OS X and who do not know much about the ‘joys’ of working on dial-up text mode terminals.

I hope this fresh new look at vi(1) will help you learn it and become more productive.

Jacek Artymiak

Chapter I

Essentials

PANIC!

When things go wrong and you are stuck feeling that vi(1) has gotten out of control, it might be a good idea (or the only way out...) to abandon all changes you have made to the files and start again. To do so, use the **:q!** ‘panic mode.’

```
casIFJK KLFkfkasgflya;ybciaqRU6gqdhxc3vT LBRC;UEN VBQWILrpc ;obFde
g\r'h \am fp'p
\ \glGwsc.c `v# $OpenBSD: sysctl.conf,v 1.46 2008/01/05 18:38:37 mbalmer Exp $
#
# This file contains a list of sysctl options the user wants set at
# boot time. See sysctl(3) and sysctl(8) for more information on
# the many available variables.
#; fu o;if 'wRenc g'dhl/sgoi:net.inet6.ip6.mforwarding=1 # 1=Permit forwarding (r
outing) of IPv6 multicast packets
#net.inet6.ip6.multipath=1      # 1=Enable IPv6 multipath routing
#net.inet6.ip6.accept_rtadv=1  # 1=Permit IPv6 autoconf (forwarding must be 0)
#net.inet.tcp.rfc1323=0        # 0=Disable TCP RFC1323 extensions (for if tcp i
s slow)
#net.inet.tcp.rfc3390=0        # 0=Disable RFC3390 for TCP window increasing
#net.inet.esp.enable=0         # 0=Disable the ESP IPsec protocol
#net.inet.ah.enable=0          # 0=Disable the AH IPsec protocol
#net.inet.esp.udpcap=0         # 0=Disable ESP-in-UDP encapsulation
#net.inet.ipcomp.enable=1      # 1=Enable the IPCOMP protocol
#net.inet.etherip.allow=1      # 1=Enable the Ethernet-over-IP protocol
#net.inet.tcp.ecn=1            # 1=Enable the TCP ECN extension
#net.inet.carp.preempt=1       # 1=Enable carp(4) preemption
#net.inet.carp.log=1           # 1=Enable logging of carp(4) packets
#ddb.panic=0                   # 0=Do not drop into ddb on a kernel panic
:q!
```

How-To:

1. Press Esc once to switch to command mode.
2. Type **:q!**
3. Press Enter/Return.

Cancelling Commands

When you want to abandon the command you were typing in command mode, press Esc twice. See also the *Undo/Redo* section on page 87.

Switching Between Command Mode and Insert Mode

Unlike the majority of text editors and word processors, vi(1) has two modes of operation: command and insert.

How you enter insert mode depends on the kind of action you want to perform. (You can learn about the different actions in *Chapter 4, Editing* on page 53.) The most important trick to master is switching back to command mode. This is actually very simple—just press Esc.

Escaping from the ex Editor Mode

For those who like to do things the old, hard way, vi(1) offers the ex(1) editor mode (a primitive text editor with no visual editing functionality). Most of us don't need it, however, and when we end up there, it's mostly by accident.

If you notice that the text you were working with has suddenly disappeared and all you can see is the : command prompt, type vi and press Enter/Return. This will get you back to vi(1).



If you want to switch to the ex(1) editor mode, type Q in command mode.

Unscrambling the Screen

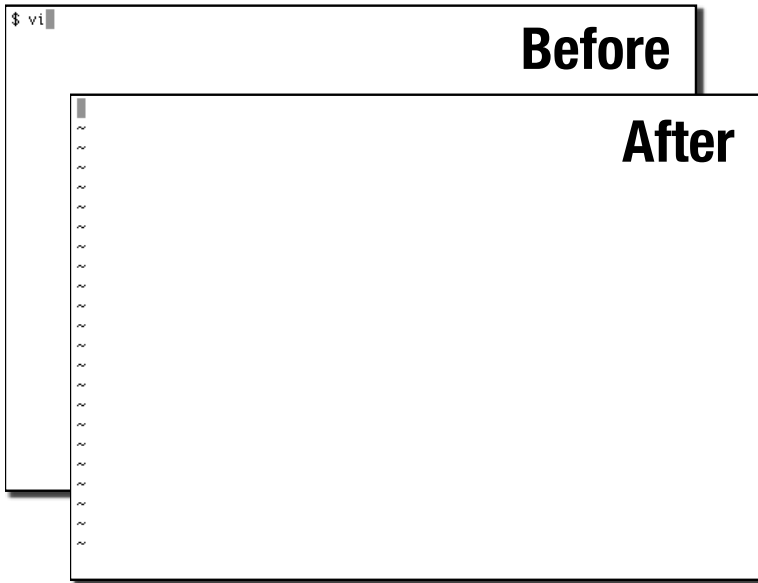
Long lines and messages sent to you by other users or the system itself may temporarily mess up the vi(1) screen. You can always clean it up with Ctrl+l (lowercase letter L) or Ctrl+r (lowercase letter R).

Chapter 2

Basic File Operations

Starting vi(1)

You can start vi(1) from the command line without telling it which file it is supposed to open. You can provide that information later (see page 10).



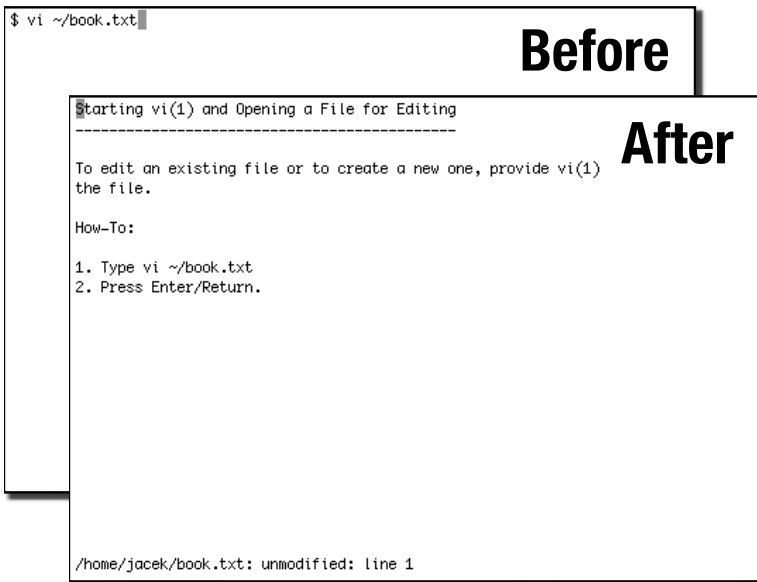
☞ The ~ (tilde) sign in the first column on every line on the vi(1) screen is an empty line marker. To tell if it is a part of the file you opened or just an empty line marker, try to move the cursor to the line with ~. If you cannot do it, it's just an empty line marker.

How-To:

1. Type vi
2. Press Enter/Return.

Starting vi(1) and Opening a File for Editing

To edit an existing file or to create a new one, provide vi(1) with the path to the file.

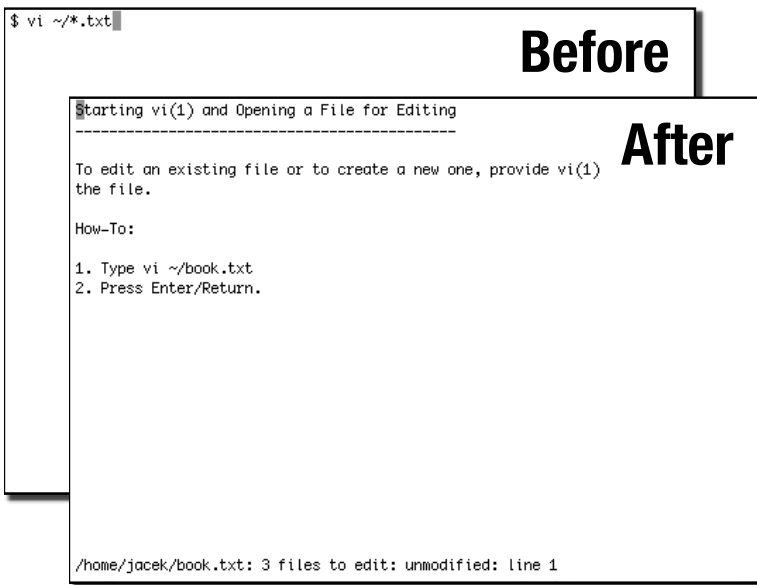


How-To:

1. Type `vi ~/book.txt`
2. Press Enter/Return.

Starting vi(1) and Opening Multiple Files for Editing

To open more than one file for editing, either list them after the `vi` command, e.g. `vi ch1.txt ch2.txt` or use one or more of the filename wildcard patterns shown in *Table 1: UNIX filename wildcards* on the next page. For example, `~/*.txt` represents all text (*.txt*) files located in your home directory.



How-To:

1. Type `vi ~/*.txt`
2. Press Enter/Return.

Table 1: UNIX filename wildcards.

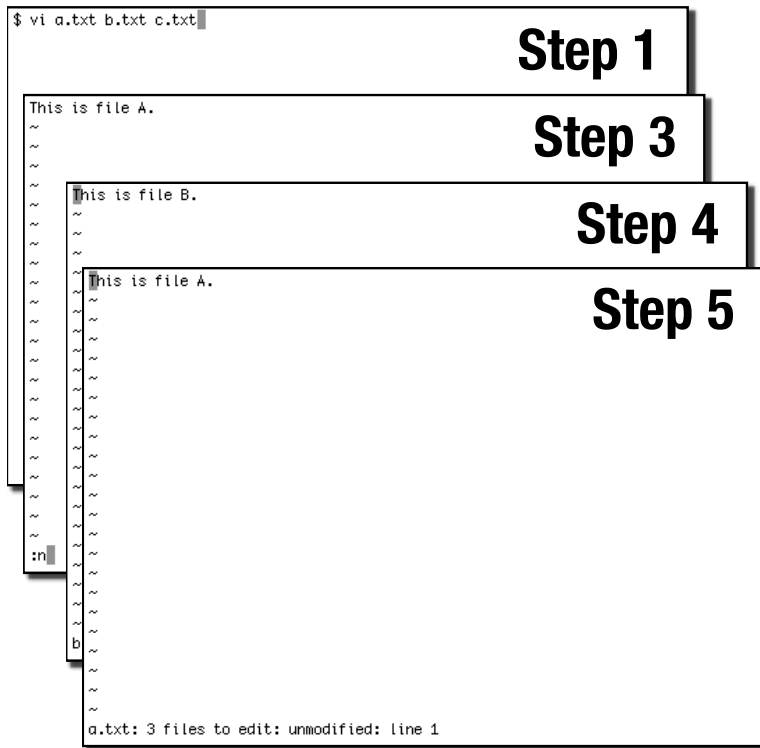
syntax	meaning	examples
<code>~</code>	home directory	<code>~</code> (your home directory) <code>~joe</code> (joe's home directory)
<code>/</code>	directory separator	<code>/home/joe/book.txt</code>
<code>*</code>	any string	<code>*.txt</code> (all filenames that end with <code>.txt</code>) <code>chapter*.txt</code> (all files whose names begin with <code>chapter</code> and end with <code>.txt</code>)
<code>?</code>	any single character	<code>chapter0?.txt</code> (all filenames that begin with <code>chapter0</code> and end with <code>.txt</code> ; the only difference between them is a single character, e.g. <code>chapter01.txt</code> , <code>chapter09.txt</code> , <code>chapter0a.txt</code> , or <code>chapter00.txt</code>)
<code>[abc]</code>	any single character from the list	<code>chapter0[123456789].txt</code> (filenames <code>chapter01.txt</code> through <code>chapter09.txt</code>)
<code>[!abc]</code>	any single character not on the list	<code>chapter0[!1234567].txt</code> (filenames outside the <code>chapter01.txt</code> through <code>chapter07.txt</code> range)
<code>[a-z]</code>	any single character from range	<code>chapter0[0-9].txt</code> (filenames <code>chapter00.txt</code> through <code>chapter09.txt</code>)
<code>[!a-z]</code>	any single character outside range	<code>chapter0[!0-9].txt</code> (filenames outside the <code>chapter00.txt</code> through <code>chapter09.txt</code> range)

☞ You can use multiple wildcards in the same filename pattern, e.g. `chapter??.*`

☞ Always test your patterns before giving them to `vi()`. One way to test is to replace `vi` with `ls`, e.g. `ls ~/*.txt` and check which files get selected.

Switching Between Files

Plain old `vi(1)` does not display all files listed on the command line after the `vi` command. Instead, it will display the first file and wait for you to do the edits. You can switch to the next file with the `:n` command. To switch to the previous file, press `Ctrl+^` (`Ctrl+Shift+6`).



How-To:

1. Type: `vi a.txt b.txt c.txt`
2. Press Enter/Return.
3. Type `:n`
4. Press Enter/Return.
5. Press `Ctrl+^`

Opening a File after Starting vi(1)

You can open a file for editing from within vi(1) with the `:e` command. The file will be placed in its own buffer, independent of other files.

✎ There is no need to close other files that you may have opened in vi(1). You can switch between them using the commands described on page 13.

✎ A buffer is just a place in memory reserved for a file.

Opening a File after Starting vi(1)

You can open a file for editing from within vi(1) with the `:e` command. The file will be placed in its own buffer, independent of other files.

There is no need to close other files that you may have opened in vi(1). You can switch between them using the commands described on page [FIXME].

How-To:

1. Press Esc to switch to command mode.
2. Type `:e ~/chapter06.txt`
3. Press Enter/Return.

Starting vi(1) and Opening a File for Editing

To edit an existing file or to create a new one, provide vi(1) with the path to the file.

How-To:

/home/jacek/chapter06.txt: unmodified: line 1

After

`:e ~/chapter06.txt`

Before

How-To:

1. Press Esc to switch to command mode.
2. Type `:e ~/chapter06.txt`
3. Press Enter/Return.

Saving the Current File

To save the current file, use the :w command. It behaves when you save a file, the current version replaces the for

How-To:

1. Press Esc to switch to command mode.

2. Type :w

3. Press Enter/Return.

Opening a File after Starting vi(1)

You can open a file for editing from within vi(1) with the :e command. The file will be placed in its own buffer, independent of other files.

There is no need to close other files that you may have opened in vi(1). You can switch between them using the commands described on page [FIXME].

How-To:

/home/jacek/chapter06.txt: 65 lines, 1392 characters.

After

How-To: W

file

You can

How-To:

:w

Before

Forcing vi(1) to Save the Current File

Sometimes vi(1) will complain that it cannot save a file that is marked as read-only. You can force vi(1) to write to the file anyway if you are logged in as the user root. To force a save, add the exclamation mark (!) after the :w command. If that doesn't work, save the file under a different name and/or location (see page 17).

Forcing vi(1) to Save the Current File

After

Sometimes vi(1) will complain that it cannot save a file only. You can force vi(1) to write to the file anyway if e user root. To force a save, add the exclamation mark (!) after the :w command. If that doesn't work, save the file under a different name and/or location (see page [FIXME]).

How-To:

1. Press Esc to switch to command mode.
2. Type :w!
3. Press Enter/Return.

Starting vi(1) and Opening a File for Editing

To edit an existing file or to create a new one, provide vi(1) with the path to the file.

How-To:

/home/jacek/book.txt: 52 lines, 1147 characters.

How-To:

:w!

Before

How-To:

1. Press Esc to switch to command mode.
2. Type :w!
3. Press Enter/Return.

Saving the Current File Under a Different Name

When you need to save the current file under a different name, use the **:w** command followed by the access path/filename.

- ✎ **vi(1)** will not change the current file's name or location. When you use **:w** alone again, files will be saved under the old name/location.
- ✎ To edit the file you just saved under a different name, open it for editing with **:e** (see page 14). The old file will still be open in **vi(1)**, but the editor will switch to the new one; now all changes saved with **:w** will be saved under the new name.

Saving the Current File Under a Different Name

When you need to save the current file under a different name followed by the access path/filename.

- **vi(1)** will not change the current file's name or location. When you use **:w** alone again, files will be saved under the old name/location.
- To edit the file you just saved under a different name, open it for editing with **:e** (see page 14). The old file will still be open in **vi(1)**, but the editor will switch to the new one; now all changes saved with **:w** will be saved under the new name.

How-To:

1. Press Esc to switch to command mode.
2. Type **:w ~/book.txt**
3. Press Enter/Return.

Forcing vi(1) to Save the Current File

/home/jacek/chapter07.txt: new file: 67 lines, 1859 characters.

After

Forcing vi(1) to Save the Current File

:w ~/chapter07.txt

Before

How-To:

1. Press Esc to switch to command mode.
2. Type **:w ~/chapter07.txt**
3. Press Enter/Return.

Saving a Part of the Current File

To save a part of the current file, use the range notation, `:n,mw!` where `n` is the number of the first line, `m` is the number of the last line, `w` stands for write, and the exclamation mark (!) forces `vi(i)` to replace the current file with the specified block.

If you forget `!`, `vi(i)` will complain and refuse to write the specified block. Rightly so, because you are *replacing* the whole file with a smaller chunk and `vi(i)` wants to make sure you know what you are doing.

Saving a Part of the Current File

To save a part of the current file, use the line range `:n,mw!` where `n` is the number of the first line, `m` is the number of the last line, `w` stands for write, and the exclamation mark (!) forces `vi(1)` to replace the current file with the specified block.

If you forget `!`, `vi(1)` will complain and refuse to write the specified block. Rightly so, because you are replacing the whole file with a smaller chunk and `vi(1)` wants to make sure you know what you are doing.

How-To:

1. Press Esc to switch to command mode.
2. Type `:4,8w!`
3. Press Enter/Return.

Saving the Current File Under a Different Name

When you need to save the current file under a different name, use the `:w` command followed by the access path/filename.

/home/jacek/book.txt: 5 lines, 485 characters.

After

When you need to save the current file under a different name, use the `:w` command followed by the access path/filename.

`:4,8w!`

Before

How-To:

1. Press Esc to switch to command mode.
2. Type `:4,8w!`
3. Press Enter/Return.

Saving a Part of the Current File Under a Different Name

Suppose you need to extract parts of the current file and save them to another file. You can do this with the help of the line range notation, `:n,mw` where `n` is the first line, `m` is the last line, and `w` stands for write. Follow this with the path and filename for your new file.

Saving a Part of the Current File Under a Different Name

Suppose you need to extract parts of the current file and save them to another file. You can do this with the help of the line range notation, `:n,mw` where `n` is the first line, `m` is the last line, and `w` stands for write. Follow this with the path and filename for your new file.

How-To:

1. Press Esc to switch to command mode.
2. Type `:4,8w ~/excerpt.txt`
3. Press Enter/Return.

Saving a Part of the Current File

To save a part of the current file, use the line range notation, `n,mw!` where `n` is the number of the first line, `m` is the number of the last line, `w` stands for write, and the exclamation mark (!) forces `vi(1)` to replace the current file with the specified block.

If you forget `!`, `vi(1)` will complain and refuse to write the specified block. `R! ~/home/jacek/excerpt.txt: new file: 5 lines, 291 characters.`

rite, and the exclamation mark (!) forces `vi(1)` to replace the specified block.

If you forget `!`, `vi(1)` will complain and refuse to write

`:4,8w ~/excerpt.txt`

After

rite, and the exclamation mark (!) forces `vi(1)` to replace the specified block.

If you forget `!`, `vi(1)` will complain and refuse to write

`:4,8w ~/excerpt.txt`

Before

How-To:

1. Press Esc to switch to command mode.
2. Type `:4,8w ~/excerpt.txt`
3. Press Enter/Return.

Appending the Current File to Another

If you want to add text from one file to the end of another, use the **:w >>** command. The current file will be appended to the end of the file that you specify.

Append the Current File to Another

If you want to add text from one file to the end of another. The current file will be appended to the end of the file.

How-To:

1. Press Esc to switch to command mode.
2. Type `:w >>~/review.txt`
3. Press Enter/Return.

█

Saving a Part of the Current File Under a Different Name

Suppose you need to extract parts of the current file and save them to another file. You can do this with the help of the line range notation, `n,mw` where `n` is the first line, `m` is the last line, and `w` stands for write. Follow this with the path and filename for your new file.

How-To:

/home/jacek/review.txt: appended: 102 lines, 3328 characters.

After

How-To:

`:w >>~/review.txt`█

Before

How-To:

1. Press Esc to switch to command mode.
2. Type `:w >>~/review.txt`
3. Press Enter/Return.

Appending a Part of the Current File to Another

Suppose you need to extract parts of the current file and add them to another. You can do this by appending blocks of lines from one file to another with the help of the line range notation `:n,mw >> filename`, where `n` is the first line, `m` is the last line, `w` stands for write, and the redirection mark (`>>`) send the selected range of lines to filename.

✎ Don't forget to use the `>>` notation.

Appending a Part of the Current File to Another

Suppose you need to extract parts of the current file and you can do this by appending blocks of lines from one file to another with the help of the line range notation `:n,mw >> filename`, where `n` is the first line, `m` is the last line, `w` stands for write, and the redirection mark (`>>`) send the selected range of lines to filename.

- Don't forget to use the `>>` notation.

How-To:

1. Press Esc to switch to command mode.
2. Type `:4,8w >>~/summary.txt`
3. Press Enter/Return.

Appending the Current File to Another

If you want to add text from one file to the end of another, use the `:w >> command`. The current file will be appended to the end of the file that you specify.

/home/jacek/summary.txt: new file: 5 lines, 403 characters.

After

If you want to add text from one file to the end of another, use the `:w >> command`. The current file will be appended to the end of the file that you specify.

`:4,8w >>~/summary.txt`

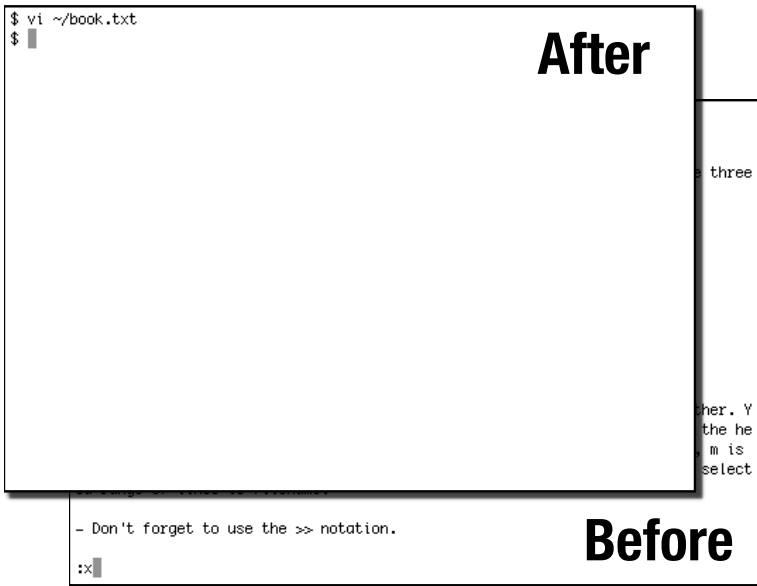
Before

How-To:

1. Press Esc to switch to command mode.
2. Type `:4,8w >>~/summary.txt`
3. Press Enter/Return.

Saving the Current File and Quitting vi(1)

When you are done editing, save the current file and quit vi(1). There are three ways to do this, all equally effective: **:x**, **:wq**, and **:ZZ**.



How-To:

1. Press Esc to switch to command mode.
2. Type **:x**
3. Press Enter/Return.

Forcing vi(1) to Save the Current File and Quit

vi(1) will not let you write to a read-only file that you may be editing, even when you are logged in as the user root. You can force it to save the file anyway with the help of the exclamation mark (:x!).



The alternative command :wq! has the same effect.

```
$ vi ~/book.txt
$
```

After

Before

How-To:

1. Press Esc to switch to command mode.

```
:x!
```

even w
anyway

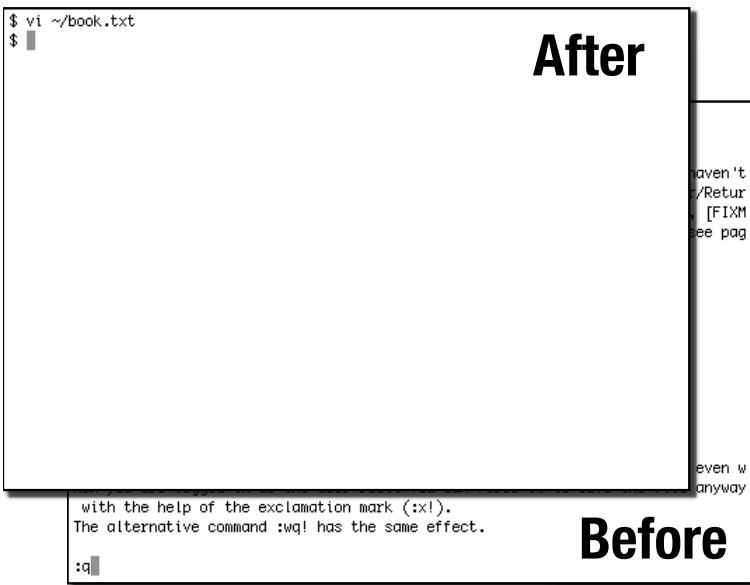
three

How-To:

1. Press Esc to switch to command mode.
2. Type :x!
3. Press Enter/Return.

Quitting vi(1) without Saving the Current File

Use the `:q` command to quit vi(1) without saving the current file. If you haven't made any changes to the file, vi(1) will exit as soon as you press Enter/Return. If you did edit it, you will have to either save it (see pages 15, 17, 22), or force vi(1) to abandon all changes since the last save (see page 25).

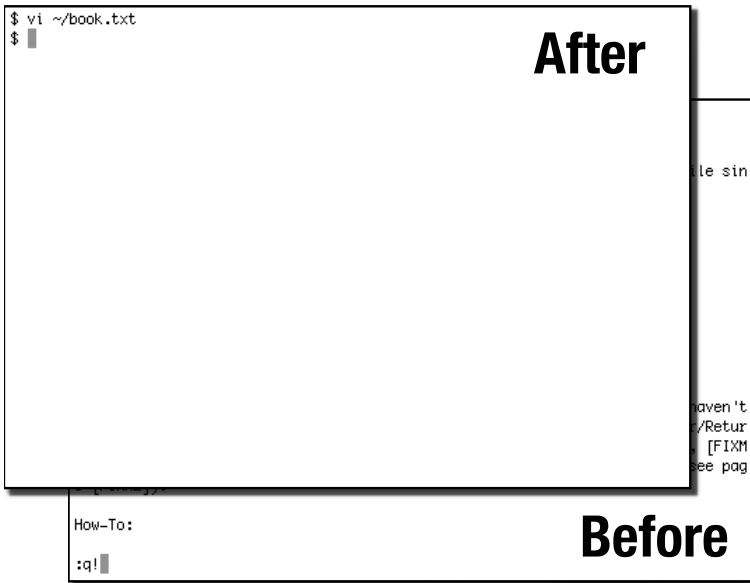


How-To:

1. Press Esc to switch to command mode.
2. Type `:q`
3. Press Enter/Return.

Forcing vi(1) to Quit without Saving the Current File

When you need to force vi(1) to abandon all changes made to the current file since the last time you saved it, use the `:q!` command.



How-To:

1. Press `Esc` to switch to command mode.
2. Type `:q!`
3. Press `Enter/Return`.

Recovering the Current File

If you decide that the changes you have made to the current file are too extensive to revert manually, use the **:e!** command. It will restore the last saved version of the current file.



You must switch vi(1) to command mode to use these commands.

Chapter 3

Cursor Movement

Moving the Cursor One Character/Line at a Time

There are two sets of keys used to move the cursor one character to the left or right, or one line up or down:



Move the cursor one character to the left: h or ←



Move the cursor one character to the right: l, space, or →

Moving the Cursor One Character/Line at a Time

Step 1

12

34

- Press j nine times to move the cursor to 1.

- Press l once to move the cursor to 2.

- Press j once to move the cursor to 4.

- Press h once to move the cursor to 3.

- Press k once to move the cursor to 1.

Step 2

12

34

- Press k once to move the cursor to 1.

Step 3

12

34

- Press k once to move the cursor to 1.

Step 4

12

34

- Press k once to move the cursor to 1.

Step 5

12

34

- Press k once to move the cursor to 1.

Step 6

12

34

- Press k once to move the cursor to 1.





Move the cursor one line down: j or ↓



Move the cursor one line up: k or ↑



Switch to the command mode to use the h, l, j, k keys when you want to move the cursor. The arrow keys can be used in either command or insert mode.

-  When the source line is shorter than the destination line, the cursor will be placed at the end of the target line.
-  When the cursor is located at the end of the source line, the cursor will be placed at the end of the target line.


How-To:

1. Open exercise file: `vi hjdkl.txt`
2. Press `j` nine times to move the cursor to 1.
3. Press `l` once to move the cursor to 2.
4. Press `j` once to move the cursor to 4.
5. Press `h` once to move the cursor to 3.
6. Press `k` once to move the cursor to 1.

Moving the Cursor *x* Characters or Lines at a Time

When moving one character or line at a time is too cumbersome, use the numeric prefix to move the cursor *x* characters or lines at a time.

You need to switch `vi(i)` to command mode and then type the number of characters or lines you want the cursor to move, followed by `h`, `j`, `k`, or `l`.

-  This technique cannot be used with the arrow keys:
← → ↓ ↑

How-To:

1. Open exercise file: `vi n-hjdkl.txt`
2. Type `10j` to move the cursor to 1.
3. Type `11l` to move the cursor to 2.
4. Type `5j` to move the cursor to 4.
5. Type `11h` to move the cursor to the first column.
6. Type `2k` to move the cursor to 3.
7. Type `3k` to move the cursor to 1.

Moving the Cursor X Characters/Lines at a Time

- Type 10j to move the cursor to 1.
- Type 11l to move the cursor to 2.
- Type 5j to move the cursor to 4.
- Type 11h to move the cursor to the first column
- Type 2k to move the cursor to 3.
- Type 3k to move the cursor to 1.

1 2

3

4

Step 1

1 2

3

4

Step 2

1 2

3

4

Step 3

1 2

3

4

Step 4

1 2

3

4

Step 5

1 2

3

4

Step 6

1 2

3

4

Step 7

Moving the Cursor to Column x

To move the cursor to a specific column, type the number of the column followed by | (vertical bar), e.g. to move to column 77, type 77|



You must switch vi(1) to command mode to use these commands.

Moving the Cursor to Column x

To move the cursor to a specific column, type the number of the column followed by | (vertical bar), e.g. to move to column 77, type 77|

Moving the Cursor to Column x

- You must switch vi(1) to command mode to use these commands.

How-To:

1. Press Esc to switch to command mode.

2. To jump to the beginning of the line, press 0

3. To jump to the end of the line, press \$

Forcing vi(1) to Quit without Saving the Current File

How-To:

1. Press Esc to switch to command mode.

2. To jump to the beginning of the line, press 0

3. To jump to the end of the line, press \$

When you need to force vi(1) to abandon all changes made to the current file since the last time you saved it, use the :q! command.

How-To:

1. Press Esc to switch to command mode.

Before

Moving the Cursor to Column x

To move the cursor to a specific column, type the number of the column followed by | (vertical bar), e.g. to move to column 77, type 77|

Moving the Cursor to Column x

- You must switch vi(1) to command mode to use these commands.

How-To:

1. Press Esc to switch to command mode.

2. To jump to the beginning of the line, press 0

3. To jump to the end of the line, press \$

Forcing vi(1) to Quit without Saving the Current File

How-To:

1. Press Esc to switch to command mode.

After

How-To:

1. Press Esc to switch to command mode.
2. To jump to the beginning of the line, press 0
3. To jump to the end of the line, press \$

Moving the Cursor to the Start or End of Line

Use 0 (zero) to move the cursor to the beginning of the line or \$ (dollar sign) to jump to the beginning or end of the line with the cursor.

Moving the Cursor to the Start or End of Line

Use 0 (zero) to move the cursor to the beginning of the line or \$ (dollar sign) to jump to the beginning or end of the line with the cursor.

Moving the Cursor to the Start or End of Line

How-To:

1. Press Esc to switch to command mode.
2. To jump to the beginning of the line, press 0
3. To jump to the end of the line, press \$

Moving the Cursor to Column x

To move the cursor to a specific column, type the number of the column followed by | (vertical bar), e.g. to move to column 77, type 77|

How-To:

1. Press Esc to switch to command mode.
2. To jump to the beginning of the line, press 0
3. To jump to the end of the line, press \$

Forcing vi(1) to Quit without Saving the Current File

When you need to force vi(1) to abandon all changes made to the current file since the last time you saved it, use the :q! command.


How-To:


1. Press Esc to switch to command mode.

Step 1

Step 2

Step 3

 You can precede \$ with the number of lines you want to jump forward, e.g. to jump to the end of the line five lines down, type 6\$ (this is correct, as the current line counts as 1).


 You must switch vi(1) to command mode to use these commands.

How-To:

1. Press **Esc** to switch to command mode.
2. To jump to the beginning of the line, press **0**
3. To jump to the end of the line, press **\$**

Moving the Cursor Between Lines


Moving the cursor between lines is accomplished with the **G** command:

 **Move the cursor to line *x*:** type the line number followed by **G**, e.g. to move to line 12, type **12G**

Alternatively, type **:*x***, e.g. **:12** moves the cursor to line 12. Note that the **:** precedes the line number.

 **Move the cursor to the first line:** type **1G** or **[[**


 **Move the cursor to the last line:** type **G** or **]]**

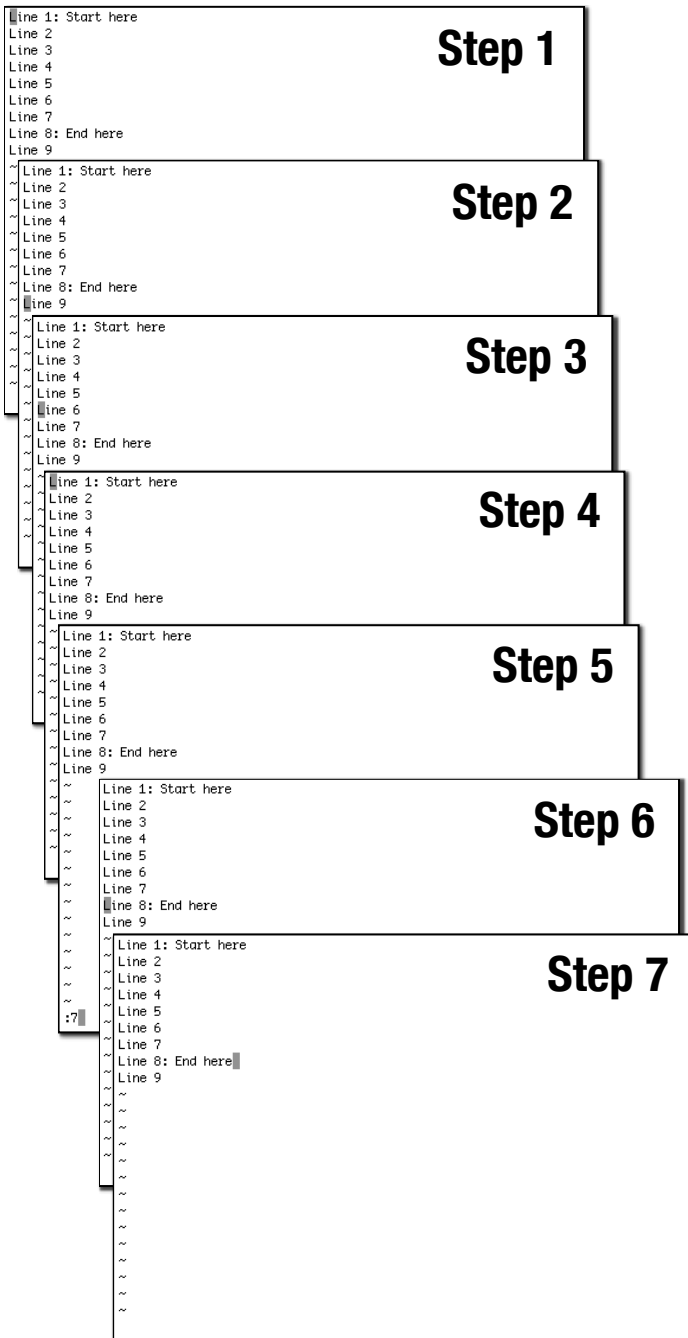
 **Move the cursor to the next line:** press **+**

 **Move the cursor *x* lines down:** type ***x*+**, e.g. to move two lines down, type **2+**

 **Move the cursor to the previous line:** press **-**

 **Move the cursor *x* lines up:** type ***x*-**, e.g. to move two lines up, type **2-**

 You must switch **vi(I)** to command mode to use the commands described in this section.



How-To:

1. Open exercise file: `vi lines.txt`
2. Type `G` to move the cursor to the last line.
3. Type `-` three times to move the cursor to line 6.
4. Type `1G` to move the cursor to the first line.
5. Type `:7` to move the cursor to line 7.
6. Type `+` to move the cursor to line 8.
7. Type `$` to move the cursor to the end of line 8.
8. Type `0` to move the cursor to the beginning of line 8.

Which Line Am I On?

If you ever need to know the current line number, try these commands:



Display the line number: type `:#`, `:num`, or `:.=`



Display the number of lines in the current file:
type `:=`



Display the line number and the total number of lines: press `Ctrl+g`



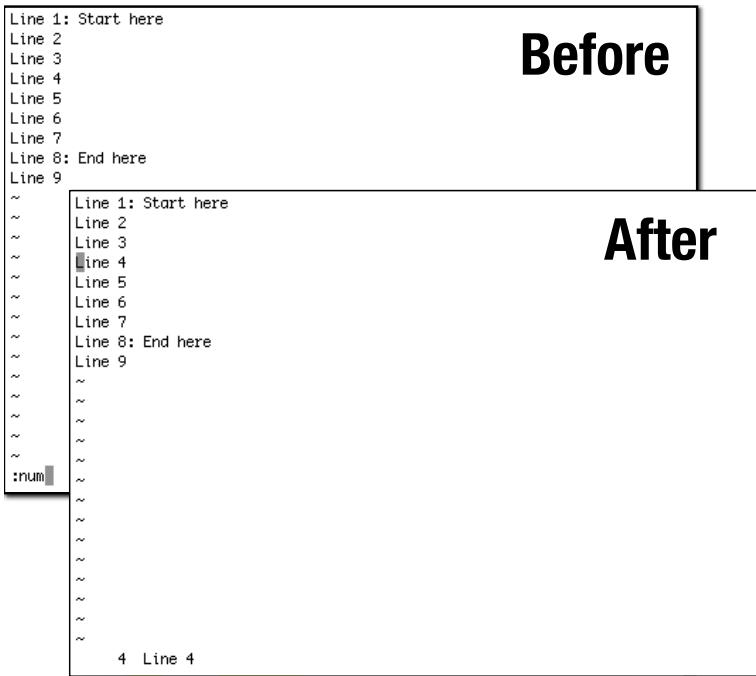
Display the number of the next line that matches a regular expression: type `:/regex/=`

The search will start on the line with the cursor and continue towards the end of the current file.

For more information about regular expressions, see page 76.



You must switch `vi(1)` to command mode to use the commands described in this section.













How-To:

1. Open exercise file: `vi line-number.txt`
2. Type `4G` to move the cursor to line 4.
3. Type `:num`
4. Press Enter/Return.

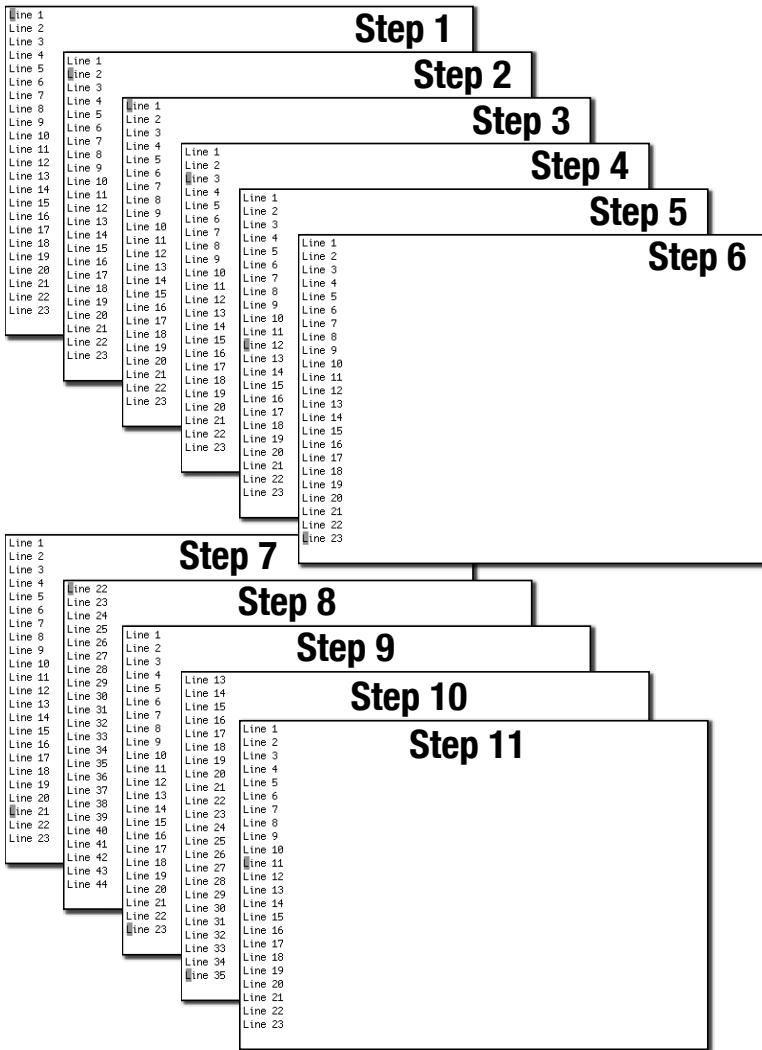
Moving the Cursor Relative to the vi(1) Screen

Moving the cursor between lines in half- and full-screen chunks is accomplished with the following commands:

-  **Move the cursor to the top of the screen:** type H
-  **Move the cursor down *x* lines relatively to the top of the screen:** type *x*H. For example, to place the cursor on the third line of the screen type 3H.
-  **Move the cursor to the middle of the screen:** type M
-  **Move the cursor to the bottom of the screen:** type L
-  **Move the cursor up *x* lines relatively to the bottom of the screen:** type *x*L. For example, to place the cursor on the third line from the bottom of the screen type 3L.
-  **Move the cursor one screen forward:** press Ctrl+f
-  **Move the cursor one screen backward:** press Ctrl+b
-  **Move the cursor half a screen down:** press Ctrl+d
-  **Move the cursor half a screen up:** press Ctrl+u
-  You must switch vi(1) to command mode to use these commands.

How-To:

1. Open exercise file: vi screen.txt
2. Type + to move the cursor from line 1 to line 2.
3. Type H to move the cursor to the top of the screen.
4. Type 3H to move the cursor three lines down from the top of the screen.
5. Type M to move the cursor to the middle of the screen (line 12).
6. Type L to move the cursor to the bottom of the screen (line 23).



7. Type 3L to move the cursor three lines up from the bottom of the screen (line 21).
8. Press Ctrl+f to move the cursor to the top of the next screen (line 22).
9. Press Ctrl+b to move the cursor to the bottom of the previous screen (line 23).

10. Press `Ctrl+d` to move the cursor to the middle of the next screen (line 35).
11. Press `Ctrl+u` twice to move the cursor to the middle of the previous screen (line 11).

Moving the Cursor to Character *x*

You can move the cursor between consecutive occurrences of characters using specialized search commands that take a single character as their argument:



Move the cursor to the next occurrence of character *x*: type `fx`. For example, to jump to the next occurrence of letter `a`, type `fa`



Move the cursor to the previous occurrence of character *x*: type `Fx`. For example, to jump to the previous occurrence of letter `a`, type `Fa`



Move the cursor to the character before the next occurrence of character *x*: type `tx`. For example, to jump to the character before the next occurrence of letter `a`, type `ta`



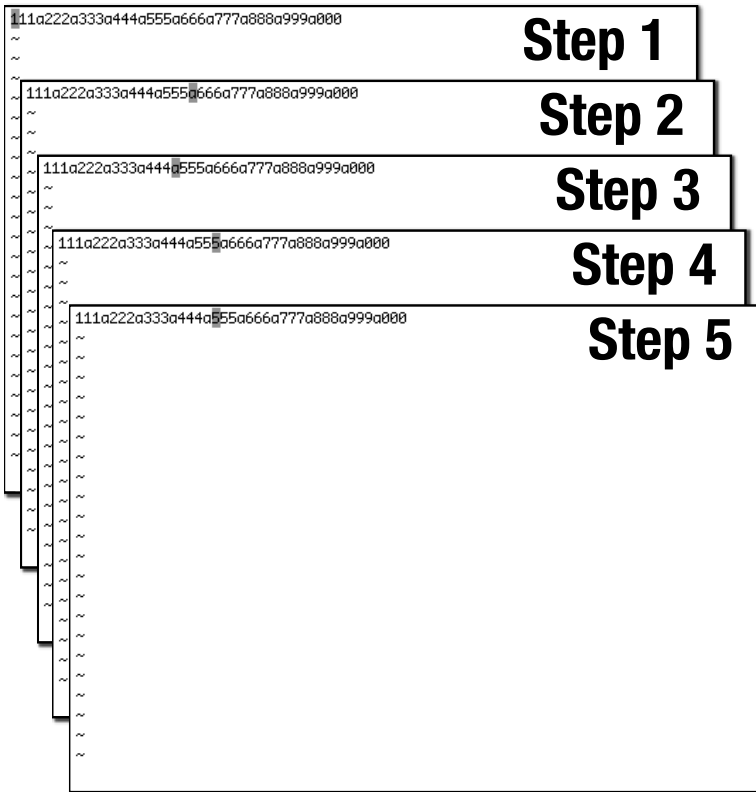
Move the cursor to the character after the previous occurrence of character *x*: type `Tx`. For example, to jump to the character after the previous occurrence of letter `a`, type `Ta`



You must switch `vi(1)` to command mode to use these commands.

How-To:

1. Open exercise file: `vi charjump.txt`
2. Type `5fa` to move the cursor to the `a` character after `555`.
3. Type `Fa` to move the cursor to the `a` character after `444`.
4. Type `ta` to move the cursor to the last `5`.
5. Type `Ta` to move the cursor to the first `5`.

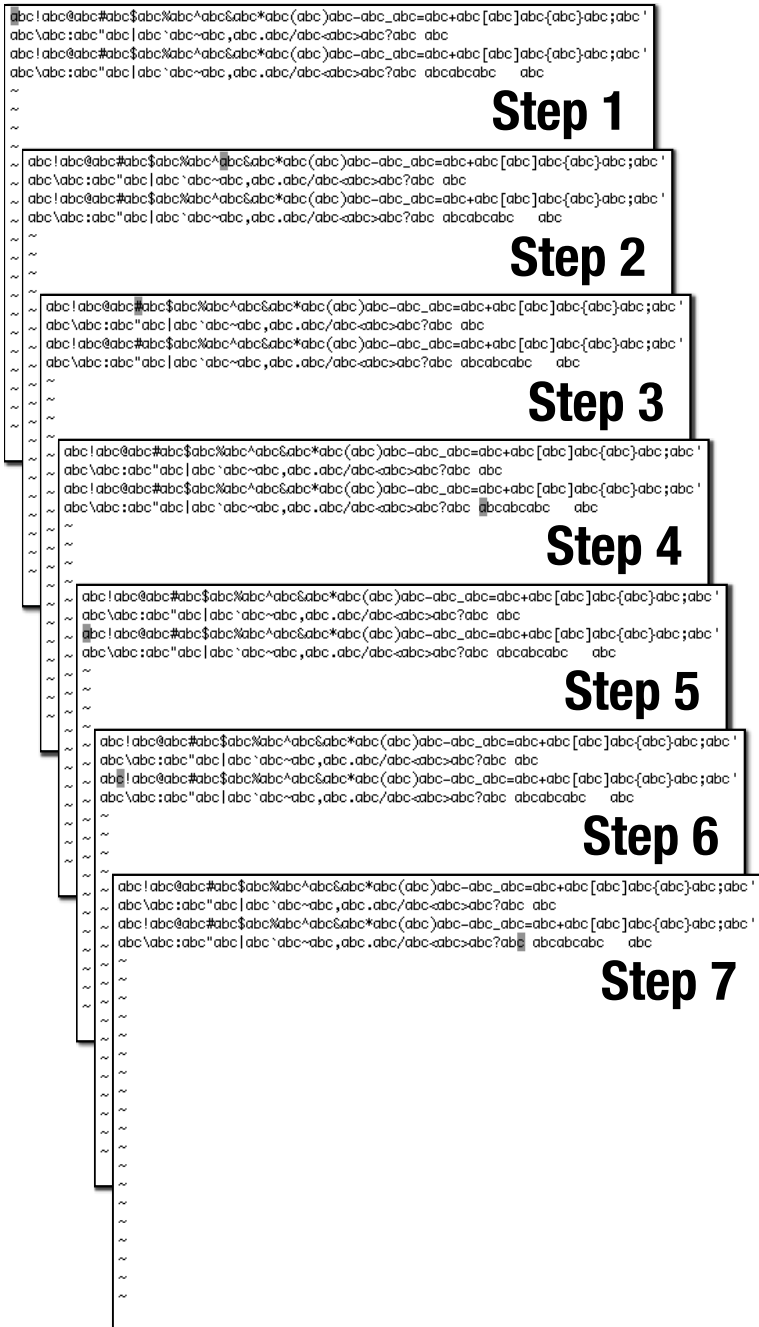



Moving the Cursor Between Words

Moving the cursor forward or backward by one or more words forward or backward is sometimes more convenient than moving it by one or more characters. There are six commands that accomplish this task. We need so many, because vi(1) defines two different types of words: strings of printable ASCII characters and strings separated by whitespace.




Strings of characters begin with a letter (a-z, A-Z), a digit (0-9), or one of the following characters: !@#\$%^&* () [] {} - = + : ; ' " \ | ` ~ , . / < > ?




 **Jump forward to the beginning of the next word:** `w`


 **Jump forward to the end of the current word:** `e`


 **Jump backward to the beginning of the previous word:** `b`


 **Strings separated by whitespace** (spaces, tabs):

 **Jump forward to the beginning of the next word:** `W`

 **Jump forward to the end of the current word:** `E`

 **Jump backward to the beginning of the previous word:** `B`

 It is possible to precede each of those commands with a number in order to move the cursor by more than one word.

 You must switch `vi(I)` to command mode to use these commands.

How-To:

1. Open exercise file: `vi words.txt`
2. Type `12w` to move the cursor to the beginning of the `abc&` string on the first line.
3. Type `7b` to move the cursor to the beginning of the `#abc` string on the first line.
4. Type `3W` to move the cursor to the beginning of the `abcbabcabc` string on the second line.
5. Type `B` to move the cursor to the beginning of the second line.
6. Type `e` to move the cursor to the end of the `abc` string at the beginning of the second line.
7. Type `E` to move the cursor to the end of the `?abc` string near the end of the second line.

Moving the Cursor Between Sentences

Moving the cursor between sentences is accomplished with the following commands:



Move the cursor to the next sentence: type)

This does not work in all implementations of vi(1), the cursor may jump to the next line instead.



Move the cursor to the previous sentence: type (



When vi(1) is asked to jump from one sentence to another, it will look for full stop (.) signs as markers.



It is possible to precede these two commands with a number in order to move the cursor by more than one sentence. For example, type 14(to move fourteen sentences backward.



You must switch vi(1) to the command mode to use these commands.

How-To:

1. Press Esc to switch to command mode.
2. To jump to the next sentence, press)
3. To jump to the previous sentence, press (

~~~~~

~ ~ ~

~

10

10

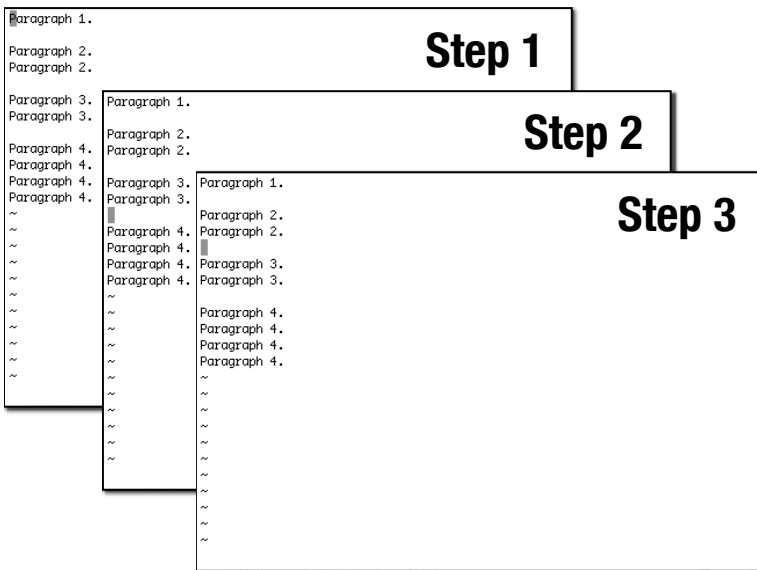
1

When `vi(i)` is told to jump from one paragraph to another, it will look for an empty line as the separator.

It is possible to precede these two commands with a number in order to move the cursor by more than one paragraph. For example, type `15}` to move fifteen paragraphs forward.



You must switch `vi(i)` to command mode to use these commands.



### How-To:

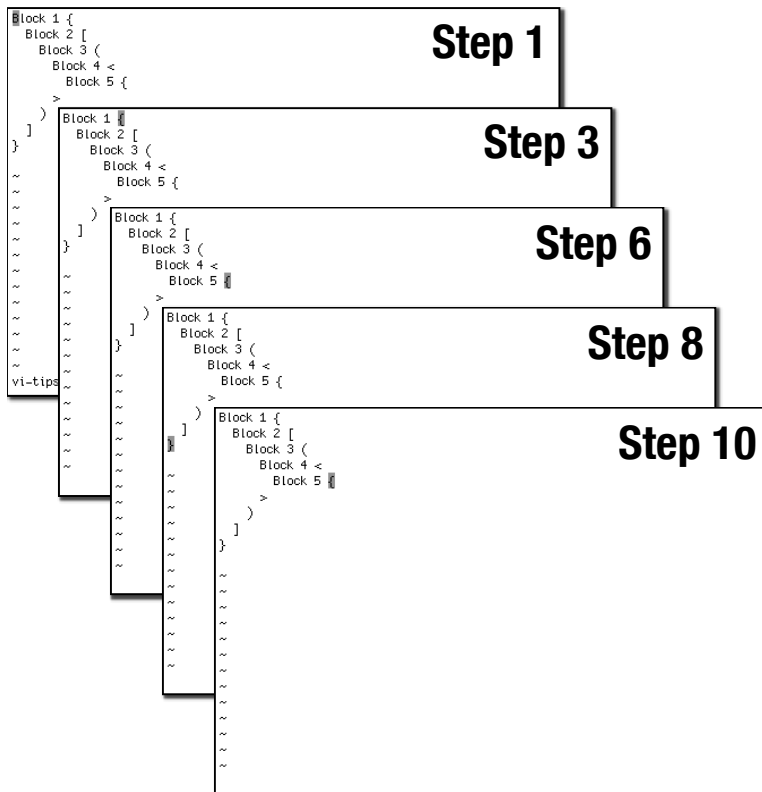
1. Press `Esc` to switch to command mode.
2. To jump to the fourth paragraph, press `}` three times.
3. To jump to the previous paragraph, press `{`

## Moving the Cursor Between Matching (), {}, [], or <>

It is easy to forget to close a block of text marked with (), {}, [], or <>. To find a matching bracket, brace, or parentheses, position the cursor over one such character and use the % command.



You must switch vi(1) to command mode to use these commands.





### How-To:

1. Open exercise file: `vi brackets.txt`
2. Press `Esc` to switch to command mode.
3. Type `$` to move the cursor to the `{` at the end of Block 1 `{` line.
4. Type `%`
5. The cursor will not move, because the curly braces `()` are not balanced.
6. Move the cursor to the `{` at the end of Block 5 `{` line.
7. Type `%`
8. The cursor will jump to the curly brace `()` on the last line.
9. Type `%` again.
10. The cursor will jump to the curly brace `()` at the end of Block 5 `{` line.

## Moving the Cursor Between Markers

Markers allow us to move between arbitrarily chosen locations inside the document. Just mark the places you want to move between and jump about as necessary:



**Add a marker:** `mx` (`x` is the single character id for the new marker)



**Jump to marker `x`:** ``x` (`x` is the single character id for the marker)



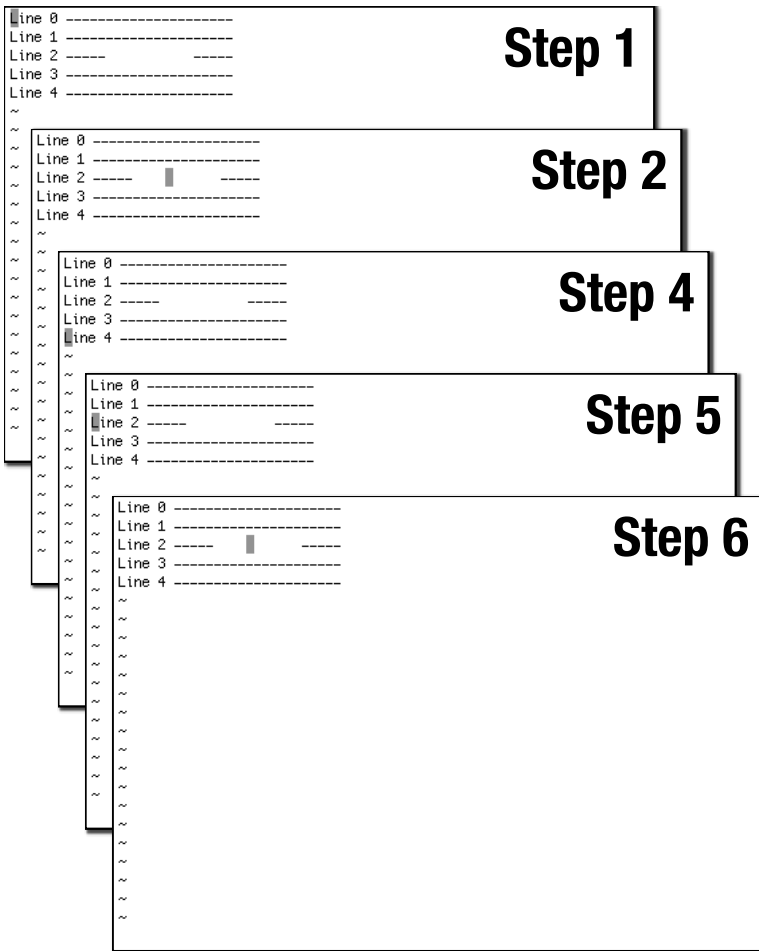
**Jump to the first non-blank character on the line with marker `x`:** `'x` (`x` is the single character id for the marker)



You can use 52 characters for markers: `a-z`, `A-Z`.



You must switch `vi()` to command mode to use these commands.



### How-To:

1. Open exercise file: `vi markers.txt`
2. Move the cursor to the free space on Line 2.
3. Type `ma` to add marker `a`.
4. Type `G` to move the cursor to the last line.
5. Type `'a` to move the cursor to the start of Line 2.
6. Type ``a` to move the cursor to marker `a`.

## Moving Around with Simple Search

When moving the cursor using other means is not convenient, there is always the ‘search for it’ option:



**Search forward:** type */string*



**Search backward:** type *?string*



**Search for the next occurrence of string in the same direction:** type *n*



**Search for the next occurrence of string in the opposite direction:** type *N*

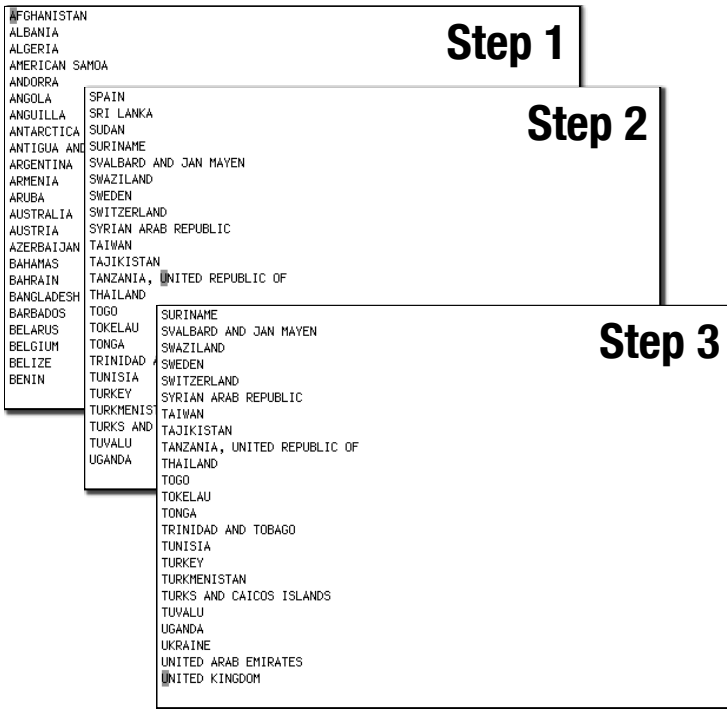
Strings that you are looking for ought to be literal strings, not regular expressions. If you want to use regular expressions in your searches, visit page 76.



You must switch vi(I) to command mode to use these commands.

### How-To:

1. Open exercise file: `vi countries.txt`
2. Type `/UNITED`
3. Hit Enter/Return
4. Press `n` twice.



## Repositioning Text Relatively to the Screen

If you would like to adjust the display by moving the text relatively to the screen, use the following commands:



**Move the line with the cursor to the top of the screen:** type `z` and press Enter/Return.



**Move the line with the cursor to the middle of the screen:** type `z`.



**Move the line with the cursor to the bottom of the screen:** type `z-`



You must switch `vi(i)` to command mode to use these commands.



## **Chapter 4**

# **Editing**



## Entering Text

Typing text into the current buffer is done using insert mode. To switch to insert mode, press **i** and type away. Whatever you type will appear before the cursor. If you want to add text after the cursor, press **a**.

Both **i** and **a** have their uppercase equivalents: **I** tells **vi(i)** to add what you type to the beginning of the line and **A** does the same at the end of the line.

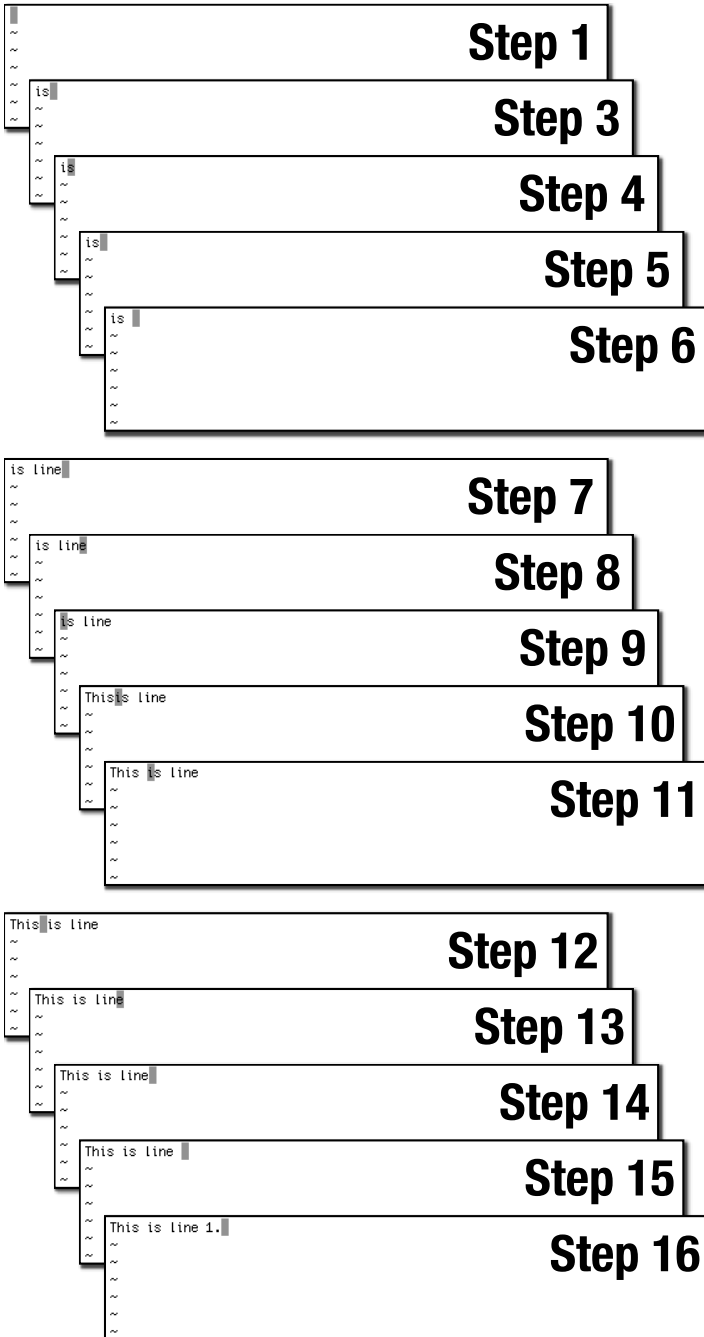
✎ Each of the **a**, **A**, **i**, **I** commands automatically switches **vi(i)** to insert mode.

✎ If you precede any of the commands with a number, whatever you type will be inserted that many times into the text you are editing.

### How-To:

1. Start **vi(i)**: **vi**
2. Type **i**
3. Type **is**
4. Press **Esc**
5. Type **a**
6. Press **Space**
7. Type **line**
8. Press **Esc**
9. Type **I**
10. Type **This**
11. Press **Space**.
12. Press **Esc**
13. Type **\$**
14. Press **a**
15. Press **Space**
16. Type **1**.
17. The end result should be **This is line 1**.





## Inserting Lines

Adding an empty line can be done in the following way: press Esc, press O, press i, press Enter/Return. That's a lot of typing. You can do it in a much easier way:



**Insert a new line below the cursor:** press o (lowercase letter O)



**Insert a new line above the cursor:** press O (uppercase letter O)

```
Line 1: Start here
Line 2
Line 3
Line 4
Line 5
Line 6
```

Step 1

```
Line 1: Start here
Line 2
Line 3
Line 4
Line 5
Line 6
```

Step 2

```
Line 1: Start here
Line 2
Line 3
Line 4
Line 5
Line 6
```

Step 4

```
Line 1: Start here
Line 2
Line 3
Line 4
Line 5
Line 6
Line 7
Line 8: End here
Line 9
```

Step 5

Both commands switch vi(1) to insert mode.



It is possible to insert more than one empty line by preceding either o or O with a number.



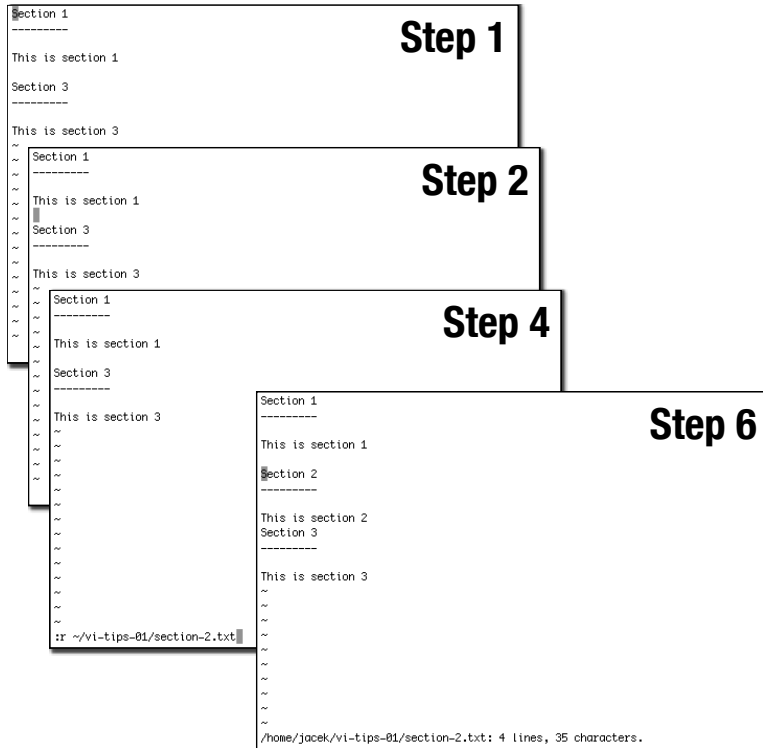
You must switch vi(1) to command mode to use these commands.

### **How-To:**

1. Open exercise file: `vi lines.txt`
2. Press O (uppercase letter O)
3. Press Esc
4. Press j
5. Press o (lowercase letter O)

## Inserting Files

To insert one file into another, use `:r` followed by the access path and filename of the file you want to insert. It will be added to the current file after the line with the cursor.



### How-To:

1. Open exercise file: `vi insert-files.txt`
2. Type `j` four times to place the cursor on the line above where you want to insert an existing file.
3. Press `Esc` to switch to command mode.
4. Type `:r ~/section-2.txt`
5. Press `Enter/Return`.
6. The file will be inserted below the line with the cursor.

# Inserting Output of a Command

To insert output generated by a script or command, use the `:r!` *command* command, which will insert the output of an external command below the current line. If you drop the `r`, the output from the external command will replace the current line.

This system's uptime is

Inserting Output from a Command

-----

This system's uptime is

Step 2

Inserting Output from a Command

-----

This system's uptime is

Step 3

Inserting Output from a Command

-----


This system's uptime is

2:01PM up 27 days, 10:11, 2 users, load averages: 0.09, 0.09, 0.08

Step 4


To insert output generated by a script or command, use the `:r!command` command, which will insert the output of an external command below the current line. If you drop the `r`, the output from the external command will replace the current line.


- Replace line *n*: type `:n!` followed by the number of the line you want to replace (numbering starts with 1), followed by `!`, and the name of the command, whose output you want to capture. For example, to replace line 7 with the output from `uptime(1)`, type `:7!uptime`
- Replace the current line: type `!.uptime`
- Replace the last line: type `:$!uptime`
- Replace the line with marker *a*: type `?'a!uptime` (for more information about markers, see page [FIXME])
- Replace a block of lines: type `:` followed by the number of the first and the last line you want to process, separated with a comma, followed by `!` and the name of the command used to process the text. For example, to replace lines 34 through 45 using output generated by `uptime(1)`, type `:34,45!uptime`

 **Replace line *n*:** type `:n!` followed by the number of the line you want to replace (numbering starts with 1), followed by `!`, and the name of the command, whose output you want to capture. For example, to replace line 7 with the output from `uptime(1)`, type `:7!uptime`

 **Replace the current line:** type `!.uptime`

 **Replace the last line:** type `:$!uptime`

 **Replace the line with marker *a*:** type `?'a!uptime` (for more information about markers, see page 48)

 **Replace a block of lines:** type `:` followed by the number of the first and the last line you want to process, separated with a comma, followed by `!` and the name of the command used to process the text. For example, to replace lines 34 through 45 using output generated by `uptime()`, type `:34,45!uptime`


 **Replace all lines:** type `:%!uptime`

#### How-To:

1. Press `Esc` to switch to command mode.
2. Place the cursor on the line above where you want to insert output from a command, e.g. `uptime()`.
3. Type `:.!uptime`
4. Press `Enter/Return`.


## Processing Text Using External Commands


You can send part or all of the current file for processing by an external command and capture the results with the `!:command` command.

 **Process line *n*:** type `:` followed by the number of the line you want to process (numbering starts with 1), followed by `!`, and the name of the command. For example, to process line 7 using `fmt()` to wrap lines on the 65th column, type `:7!fmt 65`

 **Process the current line:** type `:.!fmt 65`

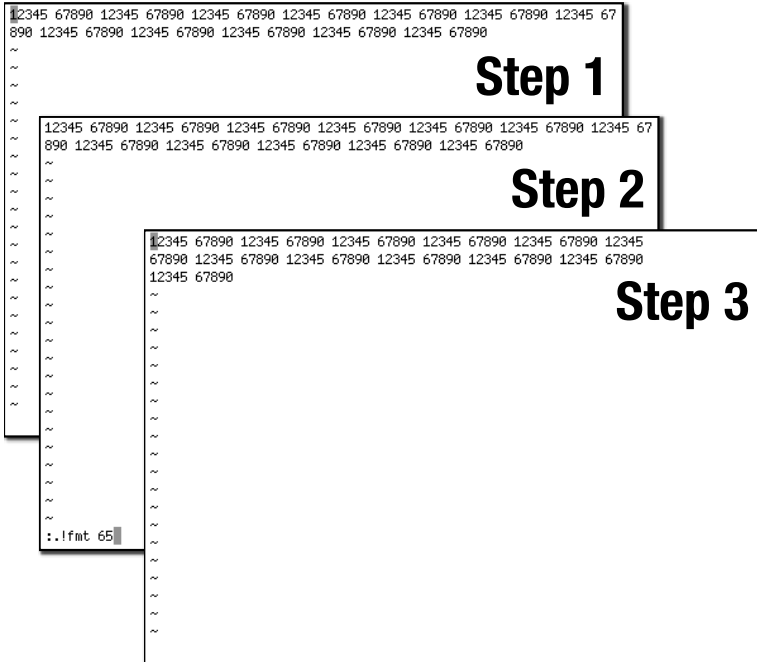
 **Process the last line:** type `:$!fmt 65`

 **Process the line with marker *a*:** type `:'a!fmt 65` (for more information about markers, see page 48)

 **Process a block of lines:** type `:` followed by the numbers of the first and the last line you want to process, separated with a comma, followed by `!` and the name of the command

used to process text. For example, to process lines 34 through 45 using `fmt(x)` to wrap lines on the 65th column, type `:34,45!fmt 65`

 **Process all lines:** type `:%!fmt 65`



**Step 1**

```
12345 67890 12345 67890 12345 67890 12345 67890 12345 67890 12345 67
890 12345 67890 12345 67890 12345 67890 12345 67890 12345 67890
```

**Step 2**

```
12345 67890 12345 67890 12345 67890 12345 67890 12345 67890 12345 67
890 12345 67890 12345 67890 12345 67890 12345 67890 12345 67890
```

**Step 3**

```
12345 67890 12345 67890 12345 67890 12345 67890 12345 67890 12345
67890 12345 67890 12345 67890 12345 67890 12345 67890 12345 67890
12345 67890
:.!fmt 65
```

### How-To:

1. Open exercise file: `vi ext-process.txt`
2. Type `:.!fmt 65`
3. Press Enter/Return

## Changing Text

The way vi(1) implements text editing functionality may catch you by surprise with its clear distinction between text entry and editing.

When you enter text, you work in insert mode, which offers very limited editing functionality. You can use the arrow keys to move back and forth, and **Backspace** or **Del** to make small changes to any text you typed since the last switch to insert mode, but that's about all you can do.

To make any serious changes you need to switch to command mode and use **c**, **C**, **cc**, **r**, or **R**.

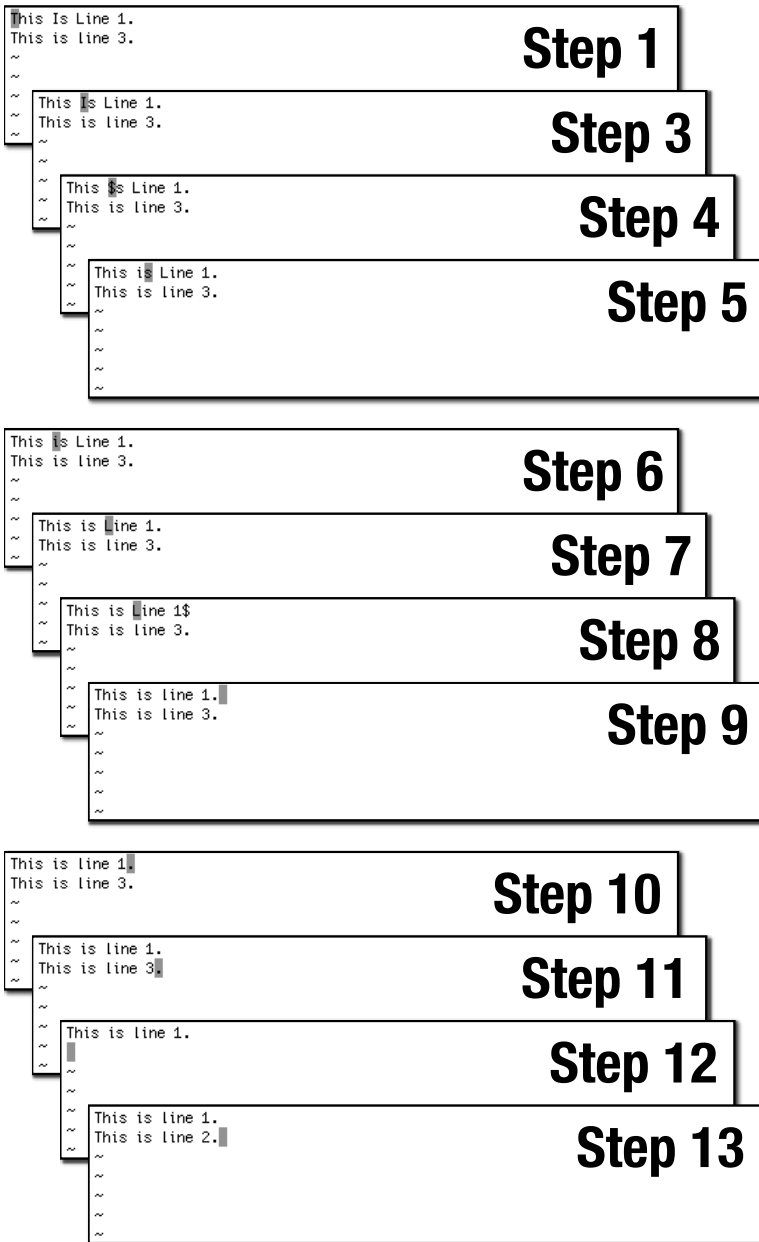
The **c** command has to be followed by one of the motion commands listed in *Table 2: Motion commands* on page 66. These commands are used to tell vi(1) how much of the existing text ought to be replaced with what you are going to type. For example, to change two words from the cursor onwards, type **c2w**. The automatically inserted **\$** character marks the end of the region of text about to be affected by your typing.

There is no need to type exactly the same number of characters to replace the previous text, vi(1) will replace the marked string with as many characters, words, or lines as you choose to type and make the necessary adjustments to the rest of the text. Should the screen become garbled in the process, **Ctrl+l** (lowercase letter **L**) will clean it up.

The other two **c** commands are handy shortcuts. To change text up to the end of line, use **C**. If **c** or **C** are not convenient enough, you can change the whole line with **cc**.

Whenever you change something using the commands discussed in this section, the old text gets placed into the default text storage register. To avoid overwriting it with the next change or cut you make, you can tell vi(1) to place the old text in one of its many named registers. To do so, precede the commands with **"x**, where **x** is the one-character register name. For example, if you want to replace two words with something else and keep the originals in register **t**, type: **"tc2w**





You can learn more about registers on page 63.



You must switch vi(1) to command mode to use these commands.

### **How-To:**

1. Open exercise file: `vi change.txt`
2. Press `Esc` to switch to command mode.
3. Type `5l`
4. Type `cl`
5. Type `i`
6. Press `Esc`
7. Type `w`
8. Type `C`
9. Type `line 1.`
10. Press `Esc`
11. Type `j`
12. Type `cc`
13. Type `This is line 2.`

Table 2: Motion commands.

| Motion                              | Command | Motion                                | Command |
|-------------------------------------|---------|---------------------------------------|---------|
| left                                | h       | right                                 | l       |
| up                                  | k       | down                                  | j       |
| next word                           | w       | previous word                         | b       |
| next word<br>(blank delimited)      | W       | previous word<br>(blank delimited)    | B       |
| end of word                         | e       | end of word<br>(blank delimited)      | E       |
| beginning of line                   | O       | end of line                           | \$      |
| first line                          | 1G      | last line                             | G       |
| line n                              | nG      | line n                                | :n      |
| next sentence                       | )       | previous sentence                     | (       |
| next paragraph                      | }       | previous paragraph                    | {       |
| next character x                    | fx      | previous character x                  | Fx      |
| forward to<br>before character<br>x | tx      | backwards to<br>before character<br>x | Tx      |
| top of screen                       | H       | middle of screen                      | M       |
| bottom of screen                    | L       |                                       |         |

## Replacing Text

When you are changing text with the ‘c’ commands, the only part of the text that gets deleted and overwritten is the part that you described using the motion commands listed in *Table 2: Motion commands* on page 66. That’s a lot of typing that not everyone wants to do. Sometimes it is more convenient to switch to ‘overtyping’ mode and replace characters as you type along.

There are two text editing commands, `r` and `R`, which come in handy when you want to replace one (`r`) or more (`R`) characters. Unlike `c` or `C`, which replace the specified amount of text with any number of characters, `r` replaces the character under the cursor and then automatically switches back to command mode, while `R` overwrites all characters under cursor until you press `Esc`.

### How-To:

1. Open exercise file: `vi replace.txt`
2. Type `w`
3. Type `c3w`
4. Type `ABCDEFG`
5. Press `Esc`
6. Type `j`
7. Type `0` (zero).
8. Type `w`
9. Type `R`
10. Type `ABCDEFG`
11. Compare lines 1 and 2

Line 1: abcdefg hijklmnop  
Line 2: abcdefg hijklmnop  
~  
~  
~

Step 1

Line 1: abcdefg hijklmnop  
Line 2: abcdefg hijklmnop  
~  
~  
~

Step 2

Line 1: abcdef\$ hijklmnop  
Line 2: abcdefg hijklmnop  
~  
~  
~  
~  
~

Step 3

Line ABCDEFGf\$ hijklmnop  
Line 2: abcdefg hijklmnop  
~  
~  
~

Step 4

Line ABCDEFG hijklmnop  
Line 2: abcdefg hijklmnop  
~  
~  
~

Step 5

Line ABCDEFG hijklmnop  
Line 2: abcdefg hijklmnop  
~  
~  
~  
~  
~

Step 6

Line ABCDEFG hijklmnop  
Line 2: abcdefg hijklmnop  
~  
~  
~

Step 7

Line ABCDEFG hijklmnop  
Line 2: abcdefg hijklmnop  
~  
~  
~

Step 8

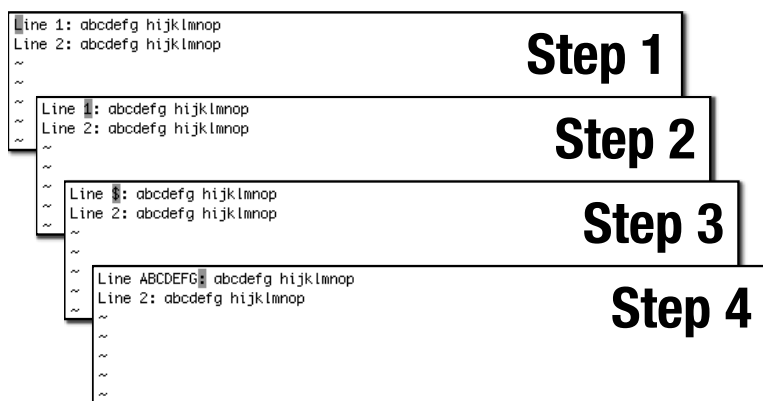
Line ABCDEFG hijklmnop  
Line ABCDEFGf\$ hijklmnop  
~  
~  
~  
~  
~

Step 10

## Replacing One or More Characters with Any Number of Characters

The **s** command lets you replace any single character with any number of characters. This command tells **vi(i)** to delete the character under the cursor and continues in insert mode until you press **Esc**.

If you precede **s** with a number, **vi(i)** will replace that many characters with whatever you type. For example, to replace 34 characters, type **34s**



When you change something using the **s** command, the old text gets placed into the default text storage register. To avoid overwriting it with the next change or cut you make, you can tell **vi(i)** to place the old text in one of its named registers. To do so, precede the **s** command with **"x**, where **x** is a one-character register name. For example, if you want to replace 34 characters with something else and keep the originals in register **t**, type: **"t34s**

You can learn more about registers on page 63.



You must switch vi(1) to command mode to use this command.

### How-To:

1. Open exercise file: `vi s-replace.txt`
2. Type `w`
3. Type `s`
4. Type `ABCDEFGG`

## Replacing the Current Line

If you would like to replace the current line with a new one in a single action, use the `S` command. The line will be deleted and vi(1) will continue in insert mode until you press `Esc`.

You can precede `S` with the number of line you want to replace. For example, to replace 72 lines, type `72S`

The diagram shows three overlapping terminal windows illustrating the steps to replace a line:

- Step 1:** The terminal shows three lines: "This is line 1.", "This is line 4.", and "This is line 3.". The cursor is at the start of the second line.
- Step 3:** The terminal shows the same three lines, but the second line is now highlighted, indicating it has been selected for replacement.
- Step 4:** The terminal shows the result after replacement: "This is line 1.", "This is line 2.", and "This is line 3.". The cursor is at the end of the second line.

Whenever you change something using `S`, the old text gets placed into the default text storage register. To avoid overwriting it with the next change or cut you make, you can tell vi(1) to place the old text in one of its many named registers. To do so, precede the `S` command with `"x`, where `x` is the one-character register name. For example, if you want to replace two words with something else and keep the originals in register `t`, type: `"t72S`

You can learn more about registers on page 63.



You must switch vi(I) to command mode to use this command.

### How-To:

1. Place the cursor on the line you want to replace.
2. Press Esc once to switch to the command mode.
3. Type S
4. Type anything you like to replace the old line.

## Deleting Text

There is a multitude of commands designed to help you delete characters, words, lines, and whole blocks of text.

### Deleting Characters:



**Delete a character under the cursor:** type x



**Delete *n* characters, starting with the one under the cursor:** type the number of characters to be deleted and then type x



**Delete a character to the left of the cursor:** press X



**Delete *n* characters to the left:** type the number of characters to be deleted and then press X



**Delete all characters, from the one under the cursor to the end of the line:** press D. An alternative to D is d\$



**Delete all characters, from the beginning of the line to the cursor:** type d0



```

Line 1: 0 1 2 3 4 5 6 7 8 9
Line 2: 0 1 2 3 4 5 6 7 8 9
Line 3: 0 1 2 3 4 5 6 7 8 9
Line 4: 0 1 2 3 4 5 6 7 8 9

```

**Step 1**

```

Line 1: 0 2 3 4 5 6 7 8 9
Line 2: 0 1 2 3 4 5 6 7 8 9
Line 3: 0 1 2 3 4 5 6 7 8 9
Line 4: 0 1 2 3 4 5 6 7 8 9

```

**Step 2**

```

Line 1: 0 2 3 4 5 6 7 8 9
Line 2: 0 1 2 3 4 5 6 7 8 9
Line 3: 0 1 2 3 4 5 6 7 8 9
Line 4: 0 1 2 3 4 5 6 7 8 9

```

**Step 3**

```

Line 1: 0 2 3 4 5 6 7 8 9
Line 2: 0 2 3 4 5 6 7 8 9
Line 3: 0 1 2 3 4 5 6 7 8 9
Line 4: 0 1 2 3 4 5 6 7 8 9

```

**Step 4**

```

Line 1: 0 2 3 4 5 6 7 8 9
Line 2: 0 2 3 4 5 6 7 8 9
Line 3: 0 1 2 3 4 5 6 7 8 9
Line 4: 0 1 2 3 4 5 6 7 8 9
Line 5: 0 1 2 3 4 5 6 7 8 9
Line 6: 0 1 2 3 4 5 6 7 8 9
Line 7: 0 1 2 3 4 5 6 7 8 9

```

**Step 5**

```

Line 1: 0 2 3 4 5 6 7 8 9
Line 2: 0 3 4 5 6 7 8 9
Line 3: 0 1 2 3 4 5 6 7 8 9
Line 4: 0 1 2 3 4 5 6 7 8 9

```

**Step 6**

```

Line 1: 0 2 3 4 5 6 7 8 9
Line 2: 0 3 4 5 6 7 8 9
Line 3: 0 1 2 3 4 5 6 7 8 9
Line 4: 0 1 2 3 4 5 6 7 8 9

```

**Step 7**

```

Line 1: 0 2 3 4 5 6 7 8 9
Line 2: 0 3 4 5 6 7 8 9
Line 3: 0 1 2 3 4 5 6 7 8 9
Line 4: 0 1 2 3 4 5 6 7 8 9

```

**Step 8**

```

Line 1: 0 2 3 4 5 6 7 8 9
Line 2: 0 3 4 5 6 7 8 9
Line 3: 0 1 2 3 4 5 6 7 8 9
Line 4: 0 1 2 3 4 5 6 7 8 9

```

**Step 9**

```

Line 1: 0 2 3 4 5 6 7 8 9
Line 2: 0 3 4 5 6 7 8 9
Line 3: 0 1 2 3 4 5 6 7 8 9
Line 4: 0 1 2 3 4 5 6 7 8 9
Line 5: 0 1 2 3 4 5 6 7 8 9
Line 6: 0 1 2 3 4 5 6 7 8 9
Line 7: 0 1 2 3 4 5 6 7 8 9

```

**Step 10**

```

Line 1: 0 2 3 4 5 6 7 8 9
Line 2: 0 3 4 5 6 7 8 9
Line 3: 0 1 2 3 4 5 6 7 8 9
Line 4: 0 1 2 3 4 5 6 7 8 9

```

**Step 11**

```

Line 1: 0 2 3 4 5 6 7 8 9
Line 2: 0 3 4 5 6 7 8 9
Line 3: 0 1 2 3 4 5 6 7 8 9
Line 4: 0 1 2 3 4 5 6 7 8 9

```

**Step 12**

```

Line 1: 0 2 3 4 5 6 7 8 9
Line 2: 0 3 4 5 6 7 8 9
Line 3: 0 1 2 3 4 5 6 7 8 9
Line 4: 0 1 2 3 4 5 6 7 8 9

```

**Step 13**

```

Line 1: 0 2 3 4 5 6 7 8 9
Line 2: 0 3 4 5 6 7 8 9
Line 3: 0 1 2 3 4 5 6 7 8 9
Line 4: 0 1 2 3 4 5 6 7 8 9

```

**Step 14**

```

Line 1: 0 2 3 4 5 6 7 8 9
Line 2: 0 3 4 5 6 7 8 9
Line 3: 0 1 2 3 4 5 6 7 8 9
Line 4: 0 1 2 3 4 5 6 7 8 9
Line 5: 0 1 2 3 4 5 6 7 8 9
Line 6: 0 1 2 3 4 5 6 7 8 9
Line 7: 0 1 2 3 4 5 6 7 8 9

```

**Step 15**

### How-To:

1. Open exercise file: `vi d-char.txt`
2. Type `10l`
3. Type `x`
4. Type `j`
5. Type `5x`
6. Type `j`
7. Type `X`
8. Type `j`
9. Type `4X`
10. Type `j`
11. Type `D`
12. Type `j`
13. Type `d$`
14. Type `j`
15. Type `d0`

### Deleting Words:



**Delete a word:** type `dw`



**Delete *n* words:** type `d` followed by the number of words to be deleted followed by `w`



**Delete the word to the left of the cursor:** type `db`

### Deleting Text Using Motion Commands:



Use motion commands to delete bigger chunks of text in a more convenient way. For example, to delete three characters to the left, type `d3h`; to remove four paragraphs, type `d4}`; etc. To see more examples of motion commands, consult *Table 2* on page 63.

### Deleting Lines:









**Delete the current line:** type `dd` or `:.d`








**Delete *n* lines:** type the number of lines to be deleted and then type `dd`


### Deleting Lines Using Ranges:


-  **Delete line *n*:** type `:n` followed by the number of the line you want to delete (numbering starts with 1), followed by `d`. For example, to delete line 7, type `:7d`
-  **Delete the current line:** type `:.d`
-  **Delete the last line:** type `:$d`
-  **Delete the line with marker *a*:** type `:'ad` (for more information about markers, see page 48)
-  **Delete a block of lines:** type `:n,m` followed by the number of the first and the last line you want to delete, separated with a comma, followed by `d`. For example, to delete lines 35 through 67, type `:35,67d`
-  **Delete all lines:** type `:%d`


### Deleting Lines Using Regular Expressions:


-  **Delete all lines matching a literal string:** type `:g/regex/d`, For example, to delete all lines that contain the word London, type `:g/London/d`
-  **Delete all lines matching a string with one variable character:** use `.` (dot). For example, to delete all lines that contain words London, london, rondon, type `:g/.ondon/d`
-  **Delete all lines matching any number of repetitions of the previous pattern:** use `*`. For example, to delete all lines that contain the words London, london, rondon, LLondon, zzzzzzzondon, ondon, etc., type `:g/.*ondon/d`
-  **Delete all lines matching any character from a set:** use `[...]`. For example, to delete all lines that contain the lowercase letters a, b, or c – e.g. Warsaw, Bombay, Chicago, but not London or Los Angeles – type `:g/[abc]/d`
-  **Delete all lines matching any character outside a set:** use `[^...]`. For example, to delete all lines that do not contain the lowercase letters a, b, or c, e.g. London, or Los


Angeles, but not Warsaw, Bombay, Chicago – type `:g/[^abc]/d`


 **Delete all lines that start with the given string:** use `^string`. For example, to delete all lines that start with the word London, type `:g/^London/d`


 **Delete all lines that end with the given string:** use `string$`. For example, to delete all lines that end with the word London, type `:g/London$/d`


 **Delete all lines that contain words that start with the given string:** use `\<string`. For example, to delete all lines that contain words that start with London (e.g. London, Londoner, Londonderry), type `:g/\<London/d`

 **Delete all lines that contain words that end with the given string:** use `string\>`. For example, to delete all lines that contain words that end with ing (e.g. running, singing, nothing), type `:g/ing\>/d`

 **Delete all lines matching any character from a range:** use `[...-...]`. For example, to delete all lines that contain the word London, but not LONDON, type `:g/[a-z]/d`

 **Delete all lines matching any character outside a range:** use `[^...-...]`. For example, to delete all lines that contain the words London and LONDON, but not london, type `:g/[^a-z]/d`

 **Delete all lines that contain \r (DOS carriage return):** use `\r`, e.g. type `:g/\r/d`

 **Delete all lines that contain \ (backslash):** use `\\`, e.g. type `:g/\\/d`

### Using Registers:

Deleted chunks of text end up in the default text storage register. To avoid overwriting it with the next change or cut you make, you can tell vi(i) to place the old text in one of its named registers. To do so, precede the whole d, D, x, or X command with "*x*", where *x* is a one-character register name. For example, if you want to delete the following two words and keep the originals in register t, type:  
`"td2w`

*Table 3: Regular expressions.*

| Pattern                              | Expression                                               | Example                                                                                                                           |
|--------------------------------------|----------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
| literal string                       | <i>literal string</i>                                    | <code>/London/</code> matches only <b>London</b>                                                                                  |
| any single character                 | <code>.</code>                                           | <code>/.ondon/</code> matches <b>London</b> , <b>london</b> , <b>rondon</b> , etc., but not <b>ondon</b>                          |
| any number of repetitions            | <code>*</code>                                           | <code>/.<b>*</b>ondon/</code> matches <b>London</b> , <b>london</b> , <b>LLondon</b> , <b>zzzzzzondon</b> , and even <b>ondon</b> |
| any character from the set           | <code>[...]</code>                                       | <code>/[abc]/</code> matches <b>Warsaw</b> , <b>Bombay</b> , <b>Chicago</b> , but not <b>London</b> or <b>Los Angeles</b>         |
| any character outside the set        | <code>[^...]</code>                                      | <code>/[^abc]/</code> matches <b>London</b> , <b>Los Angeles</b> , but not <b>Warsaw</b> , <b>Bombay</b> , or <b>Chicago</b>      |
| beginning of line                    | <code>^</code>                                           | <code>/^London/</code> matches every line that starts with <b>London</b>                                                          |
| end of line                          | <code>\$</code>                                          | <code>/London\$/</code> matches every line that ends with <b>London</b>                                                           |
| beginning of word                    | <code>\&lt;</code>                                       | <code>/\&lt;London/</code> matches every line with words that begin with <b>London</b>                                            |
| end of word                          | <code>\&gt;</code>                                       | <code>/London\&gt;/</code> matches every line with words that end with <b>London</b>                                              |
| any character from range             | <code>[...-...]</code>                                   | <code>/[a-z]/</code> matches <b>London</b> , but not <b>LONDON</b>                                                                |
| any character outside range          | <code>[^...-...]</code>                                  | <code>/[^a-z]/</code> matches <b>London</b> and <b>LONDON</b> , but not <b>london</b>                                             |
| tab, carriage return, newline        | <code>\t \r \n</code>                                    | <code>/r/</code> matches DOS-style carriage returns                                                                               |
| [ , ], \, ^, \$, .,  , ?, *, +, (, ) | <code>[ \ ] \ \ ^ \\$ \ . \   \ ? \ * \ + \ ( \ )</code> | <code>/\V/</code> matches \                                                                                                       |

## Search and Replace

Finer or global changes to text are best done with the help of regular expressions. The command used to search and replace strings is

```
:s/regex/replacement/flags
```

The `regex` pattern is one of those listed in *Table 3: Regular expressions* on page 76. The `replacement` string is the literal string you want to put in place of the regular expression you are searching for.

You can control the behavior of `:s` with two flags: `g` (replace all occurrences of `regex` on the same line without asking to confirm) and `c` (ask for permission for every match). You can use either or both flags in the same command.

If you ever need to repeat the last `:s` command, type `&`



You must switch `vi(I)` to command mode to use these commands.

### Restricting Replacements Using Ranges:



**Restrict replacements to line `n`:** type `:` followed by the number of the line you want to run `:s` on, followed by `s/regex/replacement/flags`. For example, to limit replacements to line 7, type: `:7s/london/London/g`



**Restrict replacements to the current line:** type `:` followed by `s/regex/replacement/flags`. For example, type: `:.s/london/London/g`



**Restrict replacements to the last line:** type `:` followed by `s/regex/replacement/flags`. For example, type: `:$s/london/London/g`




**Restrict replacements to the line with marker `a`:** type `:` followed by `s/regex/replacement/flags`. For example, type: `:'as/london/London/g`


- ✎ **Restrict replacements to a block of lines:** type `:` followed by the numbers of the first and the last line of the block you want to replace strings on, separated with a comma followed by `s/regex/replacement/flags`. For example, to replace strings on lines 35 through 67, type: `:35,67s/london/London/g`
- ✎ **Run search and replace on all lines,** type `:%` followed by `s/regex/replacement/flags`, e.g. `:%s/london/London/g`


### Replacing Text Using Regular Expressions:


- ✎ **Replace all literal string matches,** type `s/regex/string/g`. For example, to replace all occurrences of London with Manchester, type `s/London/Manchester/g`
- ✎ **Remove all pattern matches,** type `s/regex//g`. For example, to remove all occurrences of London, type `s/London//g`  
  
(This trick works with all regex patterns.)
- ✎ **Replace all . (dot) pattern matches:** for example, to replace all occurrences of the words London, london, rondon, etc. with LONDON, type `s/.ondon/LONDON/g`
- ✎ **Replace all matches of any number of repetitions of the previous pattern:** use `*`. For example, to replace all occurrences of the words London, london, rondon, LLondon, zzzzzzondon, ondon, etc. with LONDON, type `s/.*ondon/LONDON/g`
- ✎ **Replace all matches of any character from a set:** use `[...]`. For example, to replace letters a, b, and c with -, e.g. to change the words Warsaw, Bombay, Chicago, to W-rs-w, Bom--y, Chi--go, type `s/[abc]/-/g`
- ✎ **Replace all matches of any character outside a set:** use `[^...]`. For example, to replace all letters other than a, b, c with -, e.g. to change words Warsaw, Bombay,


Chicago, to -a--a-, ---ba-, ---ca--, type  
:s/[^abc]/-/g


 **Replace all matches of the string at the beginning of a line:** use `^string`. For example, to replace london with London on every line that starts with london, type  
:s/^london/London/g

 **Replace all matches of the string at the end of a line:** use `string$`. For example, to replace london with London on every line that ends with london, type :s/london\$/London/g


 **Replace all matches of words that start with the given string:** use `\<string`. For example, to replace all occurrences of London with Manchester at the start of similar words (e.g. London, Londoner, Londonderry), type :s/\<London/Manchester/g


 **Replace all matches of words that end with the given string:** use `string\>`. For example, to replace all occurrences of ing with ING at the end of similar words (e.g. running, singing, nothing), type :s/ing\>/ING/g

 **Replace all matches of any character from a range:** use `[...-...]`. For example, to replace all occurrences of London with L-----, type :s/[a-z]/-/g

 **Replace all matches of any character outside a range,** use `[^...-...]`. For example, to replace all occurrences of London with -ondon, type :s/[^a-z]/-/g

 **Remove all occurrences of \r (DOS carriage return),** type :s/\r\*//g

 **Replace all occurrences of \\ (backslash) with the word backslash,** type: :s/\\/backslash/g

 **Replace all lines matching a regular expression from the set that matches another regular expression,** type :g/regex1/s/regex2/string/g. For example, to replace all occurrences of York with Orleans on every line






that matches New York, type :g/New York/s/York/Orleans/g

## Cut, Copy, and Paste

Just like anything else in vi(1), the cut, copy and paste commands are implemented in a way that may surprise you. First of all, there is no single cut/copy clipboard. Second, there is no clipboard at all! What you get instead is over sixty registers. Whatever their names, they work like sixty-plus clipboards, holding any text that you change, delete, or yank (aka. copy).

vi(1) does not offer multi-level registers, but with sixty of them, that's not too much of a problem.

The following list is a short summary of pointers to the pages that contain more information:

-  To cut text, use the delete commands (see page 71) or change commands (see page 63).
-  To copy text, use the yank commands (see page 80).
-  To paste text, use the put commands (see page 84).






## Copying Text

Copying text in vi(1) is called *yanking*. Whatever the terminology the principle of operation is the same.





There is a multitude of commands designed to help you delete characters, words, lines, and whole blocks of text. You will find them listed below.

### Copying Characters:


-  **Copy a character under the cursor:** type y1 and then press either Esc or Enter/Return

-  **Copy *n* characters, starting with the one under the cursor:** type `y` followed by the number of characters to be copied followed by `1`. Then press either `Esc` or `Enter/Return`
-  **Copy a character to the left of the cursor:** type `y1h` and press either `Esc` or `Enter/Return`
-  **Copy *n* characters to the left:** type `y` followed by the number of characters to be copied followed by `h`. Then press either `Esc` or `Enter/Return`
-  **Copy all characters, from the one under the cursor to the end of the line:** type `y$` and then press either `Esc` or `Enter/Return`
-  **Copy all characters, from the beginning of the line to the character before the cursor:** type `y0` and then press either `Esc` or `Enter/Return`

### Copying Words:

-  **Copy a word:** type `yw` and then press either `Esc` or `Enter/Return`
-  **Copy *n* words:** type `y` followed by number of words to be copied, type `w`. Then press either `Esc` or `Enter/Return`
-  **Copy the word to the left of the cursor:** type `yb` and then press either `Esc` or `Enter/Return`
-  **Copy *n* words to the left of the cursor:** type `y` followed by the number of words to be copied followed by `b`. Then press either `Esc` or `Enter/Return`

### Copying Text Using Motion Commands:

-  Use motion commands to copy bigger chunks of text in a more convenient way. For example, to copy three characters to the left, type `y3h`; to copy four paragraphs, type `y4}` ; etc.

To see more examples of motion commands, consult *Table 2* on page 66.

### Copying Lines:











- ✎ **Copy the current line:** type `yy`, `:.y`, or `Y`
- ✎ **Copy *n* lines:** type the number of lines to be copied followed by `yy` or `Y`

### Copying Lines Using Ranges:

- ✎ **Copy line *n*:** type `:` followed by the number of the line you want to copy (numbering starts with 1), followed by `y`. For example, to copy line 7, type `:7y`
- ✎ **Copy the current line:** type `:.y`
- ✎ **Copy the last line:** type `:$y`
- ✎ **Copy the line with marker *a*:** type `:'ay` (for more information about markers, see page 48)
- ✎ **Copy a block of lines:** type `:` followed by the numbers of the first and last line you want to yank, separated with a comma, followed by `y`. For example, to copy lines 35 through 67, type `:35,67y`
- ✎ **Copy all lines:** type `:%y`

### Copying Lines Using Regular Expressions:

- ✎ **Copy all lines matching a literal string:** type `:g/regex/y`. For example, to copy all lines that contain the word London, type `:g/London/y`
- ✎ **Copy all lines matching a string with one variable character:** use `.` (dot). For example, to copy all lines that contain the words London, london, rondon, type `:g/.ondon/y`
- ✎ **Copy all lines matching any number of repetitions of the previous pattern:** use `*`. For example to copy all lines that contain the words London, london, rondon, LLondon, zzzzzzzondon, ondon, etc., type `:g/.*ondon/y`

-  **Copy all lines matching any character from a set:** use `[...]`. For example, to copy all lines that contain the words Warsaw, Bombay, Chicago, but not London or Los Angeles, type `:g/[abc]/y`
-  **Copy all lines matching any character outside a set:** use `[^...]`. For example, to copy all lines that contain the words London, or Los Angeles, but not Warsaw, Bombay, Chicago, type `:g/[^abc]/y`
-  **Copy all lines that start with the given string:** use `^string`. For example, to copy all lines that start with the word London, type `:g/^London/y`
-  **Copy all lines that end with the given string:** use `string$`. For example, to copy all lines that end with the word London, type `:g/London$/y`
-  **Copy all lines that contain words that start with the given string:** use `\<string`. For example, to copy all lines that contain words that start with London (e.g. London, Londoner, Londonderry), type `:g/\<London/y`
-  **Copy all lines that contain words that end with the given string:** use `string\>`. For example, to copy all lines that contain words that end with ing (e.g. running, singing, nothing), type `:g/ing\>/y`
-  **Copy all lines matching any character from a range:** use `[...-...]`. For example, to copy all lines that contain words London, but not LONDON, type `:g/[a-z]/y`
-  **Copy all lines matching any character outside a range:** use `[^...-...]`. For example, to copy all lines that contain the words London, and LONDON but not london, type `:g/[^a-z]/y`
-  **Copy all lines that contain \r (DOS carriage return):** use `\r`, e.g. type `:g/\r/y`
-  **Copy all lines that contain \ (backslash):** use `\\`, e.g. type `:g/\\/y`

### Using Registers:

Copied blocks of text end up in the default text storage register. To avoid overwriting them with the next change or cut you make, you can tell vi(1) to place the old text in one of its named registers. To do so, precede the whole y or Y command with "x, where x is a one-character register name. For example, if you want to copy the following two words and keep the originals in register t, type:

```
"ty2w
```

## Pasting Text

Cut, deleted, changed, or copied text will be stored in the default register, unless you instruct vi(1) otherwise. Pasting stored text is called *putting*, but the principles of operation are the same.

To paste text, use the p and P commands:



**Paste text after the cursor or after line:** type p



**Paste text before cursor or before the line:** type P



You must switch vi(1) to command mode to use this command.

### Using Registers:

When you copy or delete chunks of text, they end up in the default text storage register. You can change that behavior by putting them in named registers using the "x notation, where x is a one-character register name. For example, if you want to delete the two words following the cursor and keep the originals in register t, type:

```
"td2w
```

Pasting text stored in a named register is done with either "xp or "xP. If we were to continue the example, whatever was stored in register t could be pasted with "tp or "tP

## Joining Lines

To join two or more line together, use the J command:



**Join two lines:** type J



**Join more than two lines:** type the number of lines you wish to join and then type J.



You must switch vi(1) to command mode to use these commands.

**Step 1**

```
Line 1: ---
Line 2: ---
Line 3: ---
Line 4: ---
Line 5: ---
Line 6: ---
~
~
~
~
~
~
~
~
~
~
~
~
```

**Step 2**

```
Line 1: ---
Line 2: ---
Line 3: ---
Line 4: ---
Line 5: ---
Line 6: ---
~
~
~
~
~
~
~
~
~
~
~
~
```

**Step 3**

```
Line 1: ---
Line 2: --- Line 3: --- Line 4: --- Line 5: ---
Line 6: ---
~
~
~
~
~
~
~
~
~
~
~
~
```

### How-To:

1. Open exercise file: `vi join.txt`
2. Type `j` to move to the second line.
3. Type `4J`

## Changing Case

To change lowercase (a-z) to uppercase (A-Z) or vice versa, use the ~ (tilde) command.

- ✧ You can precede ~ with a number. For example, typing 4~ tells vi(i) to change the case of four characters.
  - ✧ Use motion commands to change the case of bigger chunks of text in a more convenient way. For example, to swap the case of the next three characters to the left, type ~3h; to swap the case in four paragraphs, type ~4} ; etc.
- To see more examples of motion commands, consult *Table 2* on page 66.
- ✧ You must switch vi(i) to command mode to use this command.

The screenshot shows a terminal window with a black background and white text. At the top, the prompt character is a small square, followed by the text "BcDeFgHiJkLmNoPqRsT". Below this, there are several lines of tilde characters (~). The password being entered is "AbCdEfGhIjKlMnOpQrSt", which is displayed in a lighter shade of gray. The cursor is at the end of the password string.

### How-To:

1. Open exercise file: `vi tilde.txt`
2. Type `20~` and you should see all letters change case.

## Incrementing and Decrementing Numbers

There is a quick way to increment or decrement numbers in `vi(i)`: Position the cursor on the first digit of the numeric value you want to change:



**Increment a number:** press `Ctrl+a`



**Decrement a number:** press `Ctrl+x`



You must switch `vi(i)` to command mode to use this command.

## Repeating Actions

To repeat the last action, type `.` (dot). To repeat the last action any number of times, type the number of repetitions followed by `.` (dot).



You must switch `vi(i)` to command mode to use this command.

## Undo / Redo

To undo the last action, switch to the command mode and type `u`. There is no multi-step undo/redo history in `vi(i)`, but you can revert all changes made on the current line with `U`. There's a gotcha, you can only use `U`, if the cursor is still on the line for which you want to undo changes. To redo (revert the effects of undo), type `u` or `U` again.



You must switch `vi(i)` to command mode to use this command.





## **Chapter 5**

# **Tricks**



You must switch vi(1) to command mode to use this command.

## Sending vi(1) to the Background

Another way to get out of vi(1) to do some work on the command line is to use process suspension. To send vi(1) to the background, press `Ctrl+z`, to bring it back, type `fg` on the command line.

## Shell Access

If you need to do some work on the system command line but do not want to leave vi(1) use the `:sh` command.

To go back to vi(1), type `exit` and press Enter/Return.



You must switch vi(1) to command mode to use this command.

### How-To:

1. Press `Esc` to switch to command mode.
2. Type `:sh`
3. Press Enter/Return.
4. Type `ls /bin`
5. Press Enter/Return.
6. Type `exit`
7. Press Enter/Return.





# **Index**





## A

arrow keys, 29-30, 63

ASCII, 41

## B

background processes, 92

buffers, 14, 55

current, 55

switching between, 14

## C

canceling vi(i) commands, 5

changes, abandon all, 5

changing (swapping), case, 86-87

changing, 41, 48, 63, 67, 69-70, 75,

77-78, 84, 87

characters, 41, 48

lines, 63

words, 63

clipboard, 80

command line, 9, 13, 92

command mode, 5-6, 14-26, 29-30,  
32-34, 36, 38, 40, 43-44, 46-48, 50-51,  
58-59, 61, 63, 65, 67-70-71, 77, 84-87,  
91-92

command prompt, 6

commands, external, 60-61, 91

output of,

inserting, 60-61

reading, 91

running, 91

copying, 80-84

characters, 80-81

lines, 82-83

words, 81

cursor, 9, 27, 29-30, 32-34, 36-41, 43-51,  
57, 55, 59, 61, 63, 67, 69, 71, 73, 80-81,  
84, 87

cursor movement, 9, 27, 29-30,  
32-34, 36-41, 43-51, 57, 55, 59, 61, 63,  
67, 69, 71, 73, 80-81, 84, 87

between

braces, 47-48

brackets, 47-48

markers, 48

paragraphs, 45

parentheses, 47-48

sentences, 44

words, 41

*n* characters

left, 30

right, 30

*n* lines down, 30, 34

from the top of the screen, 38

*n* lines up, 30, 34

from the bottom of the screen,  
38

one character,

left, 29

right, 29

one line,

down, 29

up, 29

one screen

backward, 38

forward, 38

relative to the screen, 38

to

character *x*,

after previous, 40

before next, 40

next, 40

previous, 40

column *n*-, 32

line,

end of, 33-34

- first, 34
- last, 34
- next, 34
- previous, 34
- start of the line, 33-34
- paragraph,
  - next, 45-46
  - previous, 45-46
- screen,
  - bottom of, 38, 51
  - middle of, 38, 51
  - top of, 38, 51
- sentence,
  - next, 44
  - previous, 44
- word,
  - end of, 43
  - next, 43
  - previous, 43
- screen, half a
  - down, 38, 40
  - up, 38, 40
- search match,
  - next, 50
  - previous, 50

cutting text, *see deleting*

## D

- deleting, 67, 69-71, 73-75, 80, 84
  - characters, 71
  - lines, 70-71, 73-75
  - words, 71, 73

display, adjusting, 51

## E

- editing, text, 10-II, 13-14, 17, 22-24, 53, 55, 63, 67, 91
- entering, text, 55, 63
- ex mode, 6

## F

file operations, 6

- filename, 11-12, 17, 59
  - changing, 17
  - wildcard, 11-12
  - testing, 11

- file, 10-II, 13-24, 26, 59
  - access path, 10, 19, 59
  - appending to, 20
    - a part of, to another, 21
  - current, 20-22, 24-26, 59
    - forcing to save and quit, 23
  - editing, 10-II
  - forced saving the current, 16
  - inserting, 59
  - location, 17
  - opening, 10, 14
    - opening multiple, 11
  - read-only, 23
  - recovery, 26
  - saving, 22
    - a part of, 18
    - under a different name, 19
  - current, 15
  - quitting, 22
  - under a different name, 17
  - switching between, 13

## I

- insert mode, 6, 29, 55, 58, 63, 69-70
- inserting, 55, 57
  - empty lines, 57
    - above the current line, 57
    - below the current line, 57
- text, 55

## J

- joining, lines, 85

## L

lines, 6, 21, 61  
     blocks of, 21, 61  
     long, 6

line, 9, 30, 36, 70  
     current, 60  
     empty, 9  
     number, 36, 70  
     source, 30  
     target, 30

lowercase, 86

## M

markers, 9, 48, 60-61, 63, 77, 82  
     ~, 9  
     end of region, 63

motion commands, 63, 66-67, 73, 81, 86  
     character *x*,  
         backward to before, 66  
         forward to before, 66  
         next, 66  
         previous, 66  
     down, 66  
     left, 66  
     line,  
         beginning of, 66  
         end of, 66  
         first, 66  
         last, 66  
         *n*.th, 66  
     paragraph,  
         next, 66  
         previous, 66  
     right, 66  
     screen,  
         bottom of, 66  
         middle of, 66  
         top of, 66  
     sentence,  
         next, 66  
         previous, 66

up, 66  
     word,  
         end of, 66  
         next, 66  
         previous, 66

## N

numbers, 87  
     decrementing, 87  
     incrementing, 87

## O

out of control, 5  
  
 overtyping mode, 67

## P

panic, 5  
  
 pasting, 80, 84  
     text,  
         after  
             current line, 84  
             cursor, 84  
         before  
             current line, 84  
             cursor, 84  
  
 process suspension, 92  
  
 processing text with external  
     commands, 61-62  
  
 putting, *see* *pasting*

## R

ranges, 18-19, 21, 61, 74, 77, 82  
     lines in,  
         first, 18-19  
         last, 18-19  
  
 redo, 87

register, 65, 69-71, 75, 80, 84  
 default, 63, 69-70, 75, 84

regular expressions, 36, 50, 74-79, 82-83

- ^, 76
- ^string, 75-76, 79, 83
- ?, 76
- ., 74, 76, 78, 82
- (, 76
- ), 76
- [^...-...], 75, 76, 79, 83
- [^...], 74-76, 78-79, 83
- [, 76
- [...-...], 75-76, 79, 83
- [...], 74, 76, 78, 83
- ], 76
- \*, 74, 76, 78, 82
- \, 76
- \\, 75, 79, 83
- \<string, 75-76, 79, 83
- \n, 76
- \r, 75-76, 79, 83
- \t, 76
- +, 76
- |, 76
- \$, 76
- any character from the given set, 74, 78, 83
- any character from the set, 76
- any character outside range, 76
- any character outside the given set, 74-75, 78-79, 83
- any character outside the set, 76
- any characters from range, 76
- any number of repetitions of the previous pattern, 74, 78, 82
- any number of repetitions, 76
- any single character (.), 74, 76, 78, 82
- backslash (\), 75, 79, 83
- beginning of line (^), 75-76, 79, 83
- beginning of word, 75-76, 79, 83
- carriage return (\r), 76

- characters in range, 75, 79, 83
- characters outside range, 75, 79, 83
- DOS carriage return, 75, 79, 83
- end of line (\$), 75-76, 79, 83
- end of word, 75-76, 79, 83
- newline (\n), 76
- literal string, 74, 76, 82
- string\>, 75-76, 79, 83
- string\$, 75-76, 79, 83
- tab (\t), 76

repeating actions, 77, 87

replacing, 60-61, 63, 67, 69-71, 77-80

- characters, 69
- lines,
  - a block of, 61
  - all, 61
  - current, 60, 70
  - last, 60
  - n.th, 60
  - old, 71
  - with marker, 60
- text, 67
  - old with new, 63
  - on all lines, 78
  - on line n., 77
  - on the current line, 77
  - on the last line, 77
  - on the line with marker, 77
  - using regular expressions, 78
  - on lines matching regular expressions, 79-80
  - within a block of lines, 78
- words, 63

root, 16, 23

## Q

quitting, see *vi*, *quitting*

## S

screen, 6, 38, 51, 63, 91  
 unscrambling, 6

scripts, 91

searching, 50, 77

backward, 50

forward, 50

shell access, 92

strings of characters, *see characters,*  
*strings of*

strings, 50, 63, 77

literal, 50, 77

with markers, 63

## T

text editor, 6

tricks, 89

## U

undo, 87

history, 87

uppercase, 86

## V

vi, 9-11, 22, 24-25, 92

forcing to quit without saving, 25

quitting without saving, 24

quitting, 22, 92

starting, 9-11

## W

way out, 5

whitespace, 43

wrapping lines, 62

## Y

yank, *see yanking*

yanking, 80