

# This is Lab Worksheet 6 - not an Assignment

This Lab Worksheet contains some practical examples that will prepare you to complete your Assignments. You do **not** have to hand in this Lab Worksheet. Make sure you complete the separate Assignments on time. Quizzes and tests may refer to work done in this Lab Worksheet; save your answers.

## 1 Before you get started - REMEMBER TO READ ALL THE WORDS

You must have an account on the Course Linux Server to do this lab. Log in to the server and use the shell. Review the Class Notes related to this worksheet as you work through it. Leave your work on the Linux server.

## The vi/vim text editor

- **vi** – console terminal-based (non-GUI) text editor, standard version (limited features)
  - **vim** – **vi** text editor - improved version (full features)
  - **gvim** – **vim** text editor - graphical version (full features) - may not be installed
  - **vimtutor** – **vim** text editor interactive tutorial - type at any shell prompt
- The Linux command **vi** is usually a link to the newer **vim** text editor. Some versions of Linux install a smaller, limited version of **vi** as the default editor and you have to explicitly ask for an upgrade to get the full **vim** version with all the great features. **gvim** (if available) starts a GUI-based version of **vim**.
  - Some distributions (e.g. Fedora 12) install a smaller, less feature-filled version of **vim** as **vi** (e.g. `/bin/vi`), but then use system aliases to alias **vi** to a larger version of **vim** (e.g. `/usr/bin/vim`), so that you never really know which version you're getting. You can define your own alias to be sure.
  - You can find out which version of **vim** you're running using the `:version` and `:help` commands. (Under Fedora, the smaller **vi** version has the wrong help files installed. Use **vim** *not* **vi** on Fedora 12.)

## vi/vim : Hard to Learn - Easy to Use

- The **vi** text editor is the **standard editor** available on most **every** Unix-derived system, including Linux, MacOSX, and BSD. It can be used on most **any terminal**, over the slowest of dial-up links. It is a pure console terminal-based program that needs **no mouse** or graphical display screen.
- Similar to learning to touch-type on a keyboard, the **vim** editor is difficult to learn but **easy** to use once you have some mastery of it. Until you master it, you will find using **vi** awkward. Once you master it, you will easily **outperform** anyone using a mouse-based editor such as Notepad. **Learn the tool!**
- The single-character command keys used in **vi** have found their way into **other programs**. The **bash** shell supports a **vi mode** for editing command lines, and the **less** and **more** pager programs (used by the **man** command) use **vi** commands to move around the screen and search for text.
- **vi** is incredibly powerful. Most students refuse to learn it well enough to get out of the **awkward** stage. They never **master** many **vi** commands. They go into **vi** insert mode and perform all text editing using the arrow and backspace keys, turning **vi** into a *slower*, mouseless version of Notepad. If you want a Unix/Linux job, learn the editor tool! If you want to use arrow keys, use the **Pico** or **Nano** editors instead.
- **vi Recommended Reference card #1** (Donald Binder):
  - Front: [http://teaching.idallen.com/cst8207/12f/notes/vi\\_refcard\\_front.pdf](http://teaching.idallen.com/cst8207/12f/notes/vi_refcard_front.pdf)
  - Back: [http://teaching.idallen.com/cst8207/12f/notes/vi\\_refcard\\_back.pdf](http://teaching.idallen.com/cst8207/12f/notes/vi_refcard_back.pdf)
- **vi Reference card #2** (Laurent Gregoire) in multiple formats and languages:
  - <http://tnerual.eriogerg.free.fr/vim.html>
- **Online interactive web tutorial** (untested): <http://lifelifehacker.com/5844890/the-interactive-vim-tutorial-teaches-you-how-to-use-the-super+efficient-vim-text-editor>

- *Vi Lovers Home Page* is <http://thomer.com/vi/vi.html>

## 1 Exercise and practice in text editing - Read All The Words

- Work in your own HOME directory. You will be creating files in your own HOME directory.
- Open the **recommended vi** reference card **before you begin**. All the commands needed for this lab are there, except for "redo last command" which is **control-R** in **vim** (not available in old versions of **vi**).
- **Before you begin**, complete the **vimtutor** tutorial program that teaches you **vim** basics.
- The following text editing exercise requires **absolute precision**. You are system technicians; small errors in configuration files can disable systems. **Accuracy is important**. You can work on each section of this exercise repeatedly until you get it letter-perfect, then move on to the next section.
- In the underlined spaces below, enter what you typed into **vi/vim** to make the given edit using the **given** number of **vi/vim** command characters (or **fewer**, if you can). Do not use any arrow keys.
- If you make a mistake, simply type the **vim undo** command character repeatedly to undo your mistake(s). You can **undo** (and **redo**) multiple times to get the file back to a state you recognize.
- You don't have to save and exit the editor to check your work, below. Use a **second terminal** so that you don't have to leave and re-enter **vim** every time. Save your work, then use the other terminal to run the **file**, **wc**, and **sum** commands shown below.
- Do **not** use insert mode and the **arrow keys** to move around the text file. Use **command mode** and the motion commands - they are much faster once you learn them. **Do not use the arrow keys!**
- When below it says "insert a word", it means the word and the **space(s)** around the word, not just the letters of the word. Make sure each word is separated from adjacent words with one space.
- Inserting text always means "insert the text then return to command mode". **Do not remain in insert mode**. Always return to command mode after an edit, so you are ready for the next command.
- Do not enter multiple spaces between words. Do not enter spaces at the start or end of lines.
  - If you want to **see** the extra spaces at the end of lines, enter: **:set list**
  - Turn off **list** view using: **:set nolist**
- Do not enter extra blank lines, especially blank lines at the bottom of the file. **No extra blank lines!**
- **Save** your work after each **successful** section, so that you can **return** to this point if you don't get the next section correct. Think of these save files as little **snapshots** of your editor session.

### 1.1 Section save1.txt

1. Create a **bash** terminal window that is at least **80** columns wide by **24** lines long. Larger is good.
2. Start the **vim** editor with this (new) file name as an argument: **lab06.txt**
3. Turn on the **vim showmode** option so you know what mode you are in **:set showmode** (This is often enabled by default.) Also useful inside **vim** is: **:set ruler**
4. Go into insert mode, enter this single line of text (mouse copy and paste) and **save** your work:  
**Royal Rhonda's repulsive, roaring rabbits ruined Randy's rutabagas**
  - You must only mouse-paste text into **vim** when in **insert** mode. Pasting into **command** mode will run your mouse-pasted text as **vim** commands! Always mouse-paste into **insert** mode!
  - **Case** matters in everything in this exercise. Use **one** space only between each word.
  - There is no final punctuation yet. Those are ASCII apostrophes, not UTF-8 smart quotes.
  - There are no leading or trailing spaces on the one line.
  - There are no leading or trailing blank lines. The file should be exactly one line long.
5. Check your work with **file**, **wc**, and **sum**. You should see these results for your one-line text file:  

```
[user@host ~]$ file lab06.txt ; wc lab06.txt ; sum lab06.txt
lab06.txt: ASCII text
 1  8 67 lab06.txt
28356      1
```
6. When you are successful, make a backup **copy** of your new one-line text file in **save1.txt**

## 1.2 Section save2.txt

1. Continue editing the **lab06.txt** file. Let's review some basic commands (see your reference card):
2. With one command keystroke, go to the **beginning** (first character) of the line. That key is:
3. Move across the line by **next words**, from left to right. That key is:
4. Move back across the line by **beginnings of words**, from right to left. That key is:
5. Move across the line by **blank-delimited words** left to right. That key is:
6. Move back across the line by **blank-delimited words** right to left. Key used:
7. Type one single character to go to the **end of the line** and **simultaneously** enter **insert** mode (one character:     ). Type a space and the word **today** after **rutabagas**. Do not add any punctuation yet. Remember to leave insert mode. **Never stay in insert mode.**
8. Type one single character to go to the **beginning of the line** and **simultaneously** enter **insert** mode (one character:     ) Type the word **Oh!** at the beginning of the line (followed by a space) before **Royal**. Remember to leave insert mode. **Never stay in insert mode.**
9. Save and check your work (two more words!) with **wc** and **sum**: **1 10 77** and **58465**
10. When you are successful, make a backup copy of your edited one-line text file in **save2.txt**

## 1.3 Section save3.txt

1. Continue editing the **lab06.txt** file.
2. Move to the start of the line (one character:     ). **Move forward** to the first upper-case **R** (two characters:     ). Efficiently **delete the word Royal** (two characters:     ).
3. **Undo** the previous deletion using one character:
4. **Redo** the previous deletion using one (control) character:
5. Add an exclamation point to the **end** of the line using only three characters, including the character used to get out of insert mode:      Remember to leave insert mode. **Never stay in insert mode.**
6. Using only four characters, duplicate the first word in the line. *Hint*: Move to the **beginning** of the line (one character:     ). Yank the **blank-delimited word** into the cut buffer (two characters     ) and **put** the word **before** the cursor position (one character:     ). Make sure you put **before** not after.
7. Save and check your work (ten words) with **wc** and **sum**: **1 10 76** and **35211**
8. When you are successful, make a copy of your edited one-line text file in **save3.txt**

## 1.4 Section save4.txt

1. Continue editing the **lab06.txt** file.
2. Turn on line numbers using: **:set number**  
The line numbers appear inside **vi/vim** only; they will **not** be saved to the edited file.
3. Using only three or four characters, duplicate the one line in the file **9** times; you should end up with **10** identical lines in the file. *Hint*: use one or two characters to yank the current line into the cut buffer:      then use a repeat count to put the buffer **after** the current line **9** times (one digit [the repeat count] and one letter:     ). Make sure you have 10 identical lines. Save the **10** identical lines and check your work with **wc** and **sum**: **10 100 760** and **52495**
4. When you are successful, make a backup copy of your edited 10-line text file in **save4.txt**
5. Continue editing the **lab06.txt** file. Using only six characters, yank **all 10** lines in the file (the whole file) then put it back **9** times, giving **100 lines** total. *Hint*: two characters to go to the beginning (first line) of the file:      two characters to yank from here to end-of-file into the cut buffer      one digit and one character to put the cut buffer **9** times **before** the top line:      You can type **control-G** to show the current file and number of lines, to confirm that the file is 100 lines. Make sure you put **before** the current line when you put.
6. Using three characters (two digits and a letter), go to line 70:
7. Using one character, move to the **top** line on your screen (the top line on your screen is **not** the same as the beginning of the file):

8. Type multiple **k** to move **straight up**, and watch the screen scroll down one line at a time.
9. Using one character, move to the **middle** line on your screen:
10. Using one character, move to the **bottom** line on your screen (the bottom line of your screen is **not** the same as the last line of the file):
11. Type multiple **j** to move **straight down**, and watch the screen scroll up one line at a time.
12. Use one character to move to the bottom line of the file (move to **end-of-file**):
13. Type three characters plus **[Enter]** to **search forward** for the two-letter pattern: **ru**
14. Type **one** character to repeat the search forward (**in the same direction**) for the next match:
15. Move to find match number 50 in the file using two digits (repeat count) and one character:
16. Type **one** character to repeat the search backward (**reverse direction**) for the previous match:
17. Undo the most recent text change(s) using the undo character:       . Your file should be back to **10** lines again. (Use **control-G** to check!) Make sure the file has **ten identical** lines before continuing.
18. **Optionally**, turn off line numbering if you don't like it (some people do): **:set nonumber**

## 1.5 Section save5.txt

1. Continue editing the 10-line **lab06.txt** file that contains 10 identical lines. (Same as **save4.txt**)
2. Go to line **8** (use one digit and one letter:       ). Move forward to the first lower-case **r** (two characters:       )
3. Now use six characters to change the word **repulsive** to **ill** *Hint*: use two command characters to change a word and enter insert mode, type three letters "ill", type one mode-change character:       . Remember to leave insert mode. Never stay in insert mode.
4. Move forward to the next lower-case **r** on this line (two characters [a repeat move is just **one** character if you know how!]:       ) and type one single character to *repeat the last text-changing command* (one character:       ). Repeating the change will also change the word **roaring** to **ill** on this line (line **8**). You can save a lot of typing using the repeat command.
5. Using only eight command characters, go to line **6** and replace the same two words with **ill**. *Hint*: go to line **6** takes two characters:        move forward to **r** takes two characters (only **one** if you know how):        *repeat the last text change* takes one character:        move forward takes two characters (only **one** if you know how):        *repeat the last text change* takes one character:
6. **Undo** the last change so that **roaring** re-appears in line **6** (one character undo:       ).
7. **Redo** the last change so that **ill** re-appears in line **6** (one control-character redo:       ).
8. **Undo** the last change so that **roaring** re-appears again in line **6** (one character undo:       ).
9. Use nine characters to go line **4** and change everything from the start of the line forward to the first comma (**inclusive**) to the word **How**. *Hint*: go to line **4** takes two characters        change text forward to comma takes three characters (a change command followed by a forward motion to a comma - it goes into insert mode)        typing **How** takes three characters and exit insert mode takes one character.
10. Save and check your work with **wc** and **sum**: **10 97 720** and **05436**
11. When you are successful, make a backup copy of your edited 10-line text file in **save5.txt**

## 1.6 Section save6.txt

1. Continue editing the **lab06.txt** file.
2. Using five characters (or less), toggle upper/lower case on every character on line two. *Hint*: go to line two takes two characters:        toggling **99** characters (more than the length of line two) takes two digits and one character:        Most of the line will now be upper-case letters. (There is also a **three**-character way to make this same change that works for **any** line length. One of the reference sheets has it.)
3. Use four characters to delete the first four blank-separated words on this line (line two). *Hint*: one character takes you to the beginning of the line        deleting four blank-separated words takes one digit and two characters:       )

4. Use four characters to replace the only lower-case letter in this line (line two) with upper-case **C** so that it says **CANDY'S** *Hint: use two characters to move forward to the **r**: \_\_\_ use two characters to replace the single **r** with **C**: \_\_\_* Make sure **all six words** on this line (line two) are now upper-case.
5. Save and check your work with **wc** and **sum: 10 93 692 and 64996**
6. When you are successful, make a backup copy of your edited 10-line text file in **save6.txt**

## 1.7 Section save7.txt

1. Continue editing the **lab06.txt** file.
2. Use four characters to go to the last line of the file (move to **end of file**) and delete everything from the comma to the end of the line. *Hint: one character to move to **end-of-file** \_\_\_ two characters to move **forward to the character** comma \_\_\_ one character to delete from the cursor **to end-of-line** \_\_\_*
3. Use three characters to make the same change to the line above (to line **9**). *Hint: one character to go **straight up** one line \_\_\_ one character to move right one character (onto the comma) \_\_\_ one character to repeat the last text change (that deletes to end-of-line) \_\_\_*
4. Use four characters to make the same change to line **7**. *Hint: two characters to go **straight up** two lines \_\_\_ one character to move right one character (onto the comma) \_\_\_ one character to repeat the last text change (that deletes to end-of-line) \_\_\_*
5. Use the same four characters to make the same change to line **5**.
6. Use two or three characters to undo the last three text changes, restoring the deleted text at the end of lines **5, 7, and 9**: \_\_\_ (You can repeat the **undo** command three times, or use a digit **3** followed by the undo command letter to repeat **undo** three times.) The text remains missing from line **10** only.
7. Save and check your work with **wc** and **sum: 10 87 643 and 48878**
8. When you are successful, make a backup copy of your edited 10-line text file in **save7.txt**

## 1.8 Section save8.txt

1. Continue editing the **lab06.txt** file.
2. Use five characters to go to line three and delete the first **69** characters, leaving only the word **today!** *Hint: two characters go to line three \_\_\_ then use two digits (the repeat count) and one character to delete **69** characters moving to the right \_\_\_*
3. You are currently on line three. Insert the word **Not** (and a space) at the start of line three, in front of **today!** Remember to leave insert mode. **Never stay in insert mode.**
4. Use three characters to exchange lines three and four. *Hint: two characters delete the current line three (into the cut buffer; line four moves up and becomes the new current line three) \_\_\_ one character puts the cut buffer **after** the current line \_\_\_* Line four, containing the contents of the put cut buffer, becomes the current line: **Not today!**
5. Starting where you are on line four, use two characters to join line four onto the end of line three. *Hint: use one character to move **straight up** to line three and one character to **join** the two lines: \_\_\_*
6. Move line one down to be line three, so that all the lines starting with **Oh!** are together. *Hint: move to line one; delete line one into the cut buffer; move down one line (to line two) and put the cut buffer **after** the current line. Line one is now line three. All lines starting with **Oh!** are together.*
7. Move to the start of line three, the first line starting with **Oh!**
8. Use two characters to delete the first word (including the punctuation) on this line (line three): \_\_\_
9. Use three characters to move to the next line (line four) and delete the first **two** words. *Hint: one character moves **straight down** to the same place on the line below \_\_\_ and you can repeat the last text change (delete a word) twice using a digit followed by one character to repeat the last text-changing command \_\_\_*
10. Use just two characters to delete the first **two** words on the next line (line five) \_\_\_. *Hint: one character moves **straight down** to the same place on the line below and one character repeats the last text change, which was to delete **two** words.*

11. Use just three characters to skip a line and delete the first two words on the next line (line seven). Hint: two characters move **straight down** two lines `____` one character repeats the last change (which was to delete two words) `____` You can save a lot of typing using the *repeat last change* command.
12. Save and check your work with **wc** and **sum: 9 72 551 and 54341**
13. When you are successful, make a backup copy of your edited 9-line text file in **save8.txt**

## 1.9 Verification Section

You have created one **lab06.txt** file and eight **save[1-8].txt** files. Execute the following shell command lines to record your work. Make sure you get the typing **exactly** correct in these lines:

```
mkdir lab06
mv lab06.txt save?.txt lab06/
ls -dils lab06/* >lab06/.verify.txt
sum lab06/* >>lab06/.verify.txt
```

Note the use of a *hidden* file name for the output redirection. This hidden name will not be matched by the shell GLOB patterns.

Check to ensure that you have 9+9=18 lines of output in the **.verify.txt** file. (Recall the command that counts lines inside a file. Use it to count the lines in your **.verify.txt** file. There should be 18 lines.)

Compare the output of the above **ls -dils** command (without redirection) with the output of this command:

```
find lab06 -ls
```

You will recall that the man page for **find** documents that the **find -ls** option produces "**ls -dils**" format output, and the above shows that this is true.

The next page contains some optional advanced editing features.

## 2 Advanced Editing Section (optional)

- This section shows off two of the more advanced and cool **vi/vim** editor features.
- This section is optional. I hope it interests you in learning more about **vi/vim**
- Copy the **lab06.txt** file to **adv10.txt** before you begin. Do not edit the wrong file.

### 2.1 Send any number of lines to an external command

You can send any number of lines from your editor buffer **into the standard input** of any command. The command may process those lines and the **output** of the command will **replace the lines** in the editor buffer.

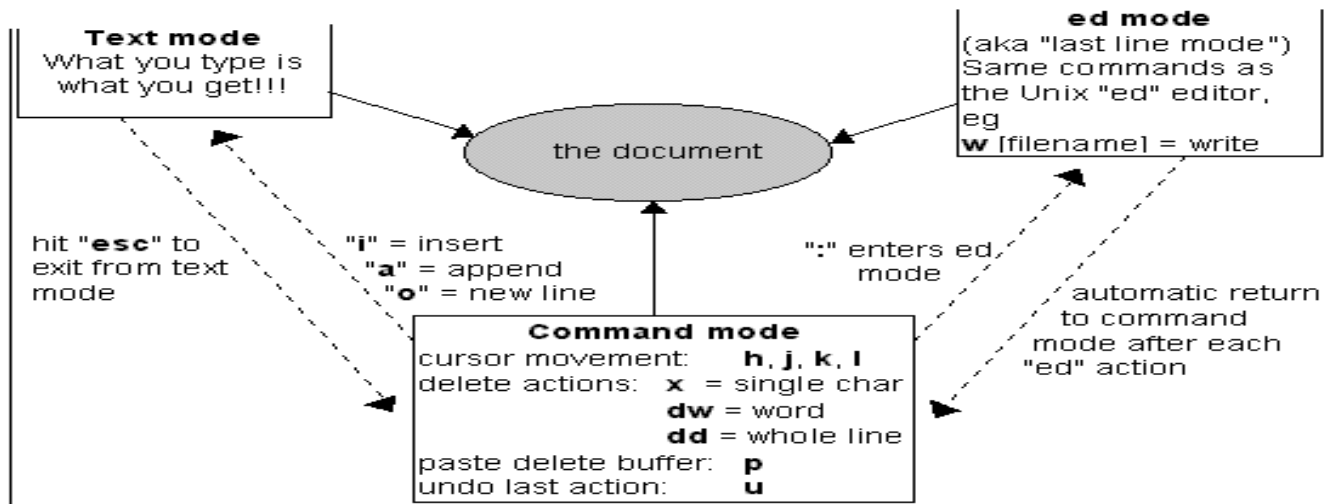
1. Edit the **adv10.txt** file (a copy of **lab06.txt**) that contains nine lines.
2. Go to the first line of the file and type this four-character command followed by **[Enter]**: **!Gwc ↵**
  - this sends all lines from the cursor to end-of-file into **wc** and replaces them with the output of **wc**
  - you should see the expected **9 72 551** output from **wc**
3. Undo the last change. (The output from **wc** disappears and the nine original lines return.)
4. Go to the first line of the file and type this five-character command followed by **[Enter]**: **!Gsum ↵**
  - sends all lines from the cursor to end-of-file into **sum** and replaces them with the output of **sum**
  - you should see the expected **54341** output from **sum**
5. Undo the last change. (The output from **sum** disappears and the nine original lines return.)
6. Go to the last line of the file (line nine) and **open** a new blank line (line ten).
7. On the new blank line type this text: **ls -li lab06.txt save?.txt**
8. Send the current line into **bash** by typing this seven-character command: **!!bash ↵**
  - when you push **[Enter]** this sends just the current line (line ten) into the **bash** shell program
  - **bash** executes the commands it is reading on standard input and the **output** goes into the editor
  - you should see the multi-line output of the **ls** command replacing line ten
9. Go to line one and execute the seven-character command: **!Gsort ↵** What happens?
10. Go to line five and execute: **!!date ↵** What happens? Try other commands!
11. Quit the editor without saving anything.

### 2.2 Record any sequence of edit commands in a macro

You can **record** any complex sequence of editor commands into a single-letter macro that you can **re-execute**.

1. Edit the **adv10.txt** file (a copy of **lab06.txt**) that contains nine lines.
2. Go to the beginning of the top line of the file.
3. Record a macro named **q** by typing **qq** that will turn on the **recording** flag in the bottom line **status bar**. You are now **recording** all the **commands** you are using into buffer **q**. Do these commands:
  - Move forward in this line to the fourth space: **4f** (*there is a space after the **f** command*)
  - Append three asterisks and a space after the cursor (and always exit insert mode): **a\*\*\* ESC**
  - Close the editing macro by typing a single **q** that turns **off** the recording flag. Macro **q** is ready.
4. The first line now has three asterisks in it: **... CANDY'S \*\*\* RUTABAGAS ...**
5. Move to the beginning of any other line in the document and re-execute this macro by typing: **@q**  
Note how the macro executes the same editor commands and makes the same change to this line.
6. Move to the **second** space in any line in the document and re-execute the macro: **@q**  
Note how the macro makes the same change to the **sixth** space, because it moves forward four spaces.
7. Go to the last line of the file (line nine) and **open** a new blank line (line ten).
8. On this blank line dump the macro contents by typing: **"qp** (*that is a **single double quote** to start*)
9. Change the three asterisks to **Fancy Macro** and re-save the macro: **"qyy** (*one double quote*)
10. Go to any line and try the new macro. You can dump and modify any macro after you have recorded it.

# Yet Another vi/vim Reference Card (not recommended)



## 1 Adding new text (switching from command mode into insert mode)

<b>i</b>	insert new text in front of cursor	<b>o</b>	(lowerc) open new line after (below) cursor
<b>a</b>	insert new text after cursor	<b>O</b>	(upperc) open new line before (above) cursor
<b>A</b>	insert new text at the <b>end</b> of the line	<b>I</b>	insert new text at the <b>start</b> of the line

## 2 Moving around without the arrow keys (using command mode)

<b>k</b>	move straight up (backwards) to the same place one line above	<b>l</b>	(lower case <b>L</b> ) move right (forward) one char
<b>h</b>	move left (backward) one character	<b>w</b>	move right (forward) one word
<b>b</b>	move left (backward) one word	<b>j</b>	move straight down (forward) to the same place one line below
<b>↓</b>	( <b>ENTER</b> ) move down (forward) one line and go to start of line	<b>^</b>	move to the start of the text in a line
<b>^</b>	move to the start of the text in a line	<b>\$</b>	move to the end of a line

## 3 Editing text (using command mode)

<b>x</b>	delete one character (under cursor)	<b>dd</b>	cut (delete) entire current line into cut buffer
<b>r</b>	replace character (under cursor)	<b>yy</b>	copy (yank) current line into cut buffer
<b>dw</b>	delete one word (containing cursor)	<b>p</b>	(lowercase) put cut/copied text <b>after</b>
<b>cw</b>	change word (from cursor)	<b>P</b>	(uppercase) put cut/copied text <b>before</b>
<b>J</b>	join the next line to the end of this line	<b>u</b>	<b>^R</b> <b>undo</b> and <b>redo</b> last editing command

## 4 Search and Replace

<b>/text</b>	search forward for next "text"	<b>?text</b>	search backwards for previous "text"
<b>:s/old/new/</b>	replace <b>old</b> with <b>new</b> once	<b>n N</b>	search forwards, backwards for same text
<b>:s/old/new/g</b>	replace all occurrences of <b>old</b> with <b>new</b> on current line only	<b>:1,\$s/old/new/g</b>	replace all occurrences of <b>old</b> with <b>new</b> on all lines from <b>1</b> to <b>\$</b> (end)

## 5 Save and/or Exit (also use upper-case **zz** in command mode)

<b>:w filename</b>	write as <b>filename</b>	<b>:q</b>	quit, exit, leave <b>vi</b>
<b>:w</b>	write using existing filename	<b>:q!</b>	<b>quit, discarding changes</b>