The vi editor

General

- `vi`*sual editor*
  - Provides a full-screen interface to the underlying editor `ex`
- Pronounced as *vee-eye*
- One of the major contributions of BSD
- Written by Bill Joy
- Primary text editor available on all versions of Unix
  - Not as friendly as some of the modern mouse-based editors but is very powerful once you have mastered it (true to Unix tradition)
- Used to create new files or edit existing ones
  - Can be used as a *very simple* word processor
  - Only supports simple text processing; you may have to use a text-processing package such as `nroff`, `troff`, LaTeX, Interleaf, or FrameMaker for complex text processing
- Being replaced by `vim` which retains its basic commands

Command syntax

- The editor is invoked by

```
$ vi filename
```

  - `filename` is optional, and if supplied, indicates the name of the file to be created or edited
  - More than one file name can be supplied at the command line, and `vi` will allow you to edit all those files one after another
  - If `filename` does not exist, it is created when you indicate to `vi` to save any changes
  - By default, the cursor is positioned in the top left corner of the screen (the first character in the file)
- File to be edited is copied into an internal system buffer; saving the edits overwrites the file on the disk
- If the first column in the screen is nothing but tildes (~ character), it indicates that the file is empty; you may also get an indication about the file being new in the last line of the screen

vi modes

- Modes are extremely important in `vi`

- There are only two modes
  1. Command mode
     - Default mode when you start `vi`
     - Allows you to issue all sorts of commands, including the commands to `ex` and shell
     - Allows you to move the cursor, either on-screen or to search for some pattern
     - To verify that you are in command mode, press the ESC key
       * A beep confirms that you are in command mode
       * No beep puts you into command mode
  2. Insert mode
     - Allows you to insert text
     - Whatever you type gets added into the file, just like a regular word processor except that you cannot move the cursor at will

Command mode

- Cursor movement commands
  - These commands allow you to position the cursor anywhere on the screen, or move to a different screen
  - The most basic commands are

Key	Effect
`h`	Move cursor one character to the left
`j`	Move cursor one line down (same column)
`k`	Move cursor one line up (same column)
`l`	Move cursor one character to the right

  - These commands have been listed in the order in which they appear on the keyboard
  - The commands are relics from the era when the terminal keyboards did not necessarily have the arrow keys, and are guaranteed to work if the arrow keys do not work properly
  - These keys also allow you to move the cursor without having to lift your fingers from the central (main) keyboard
  - If you attempt to move in a direction where there in no text, you will hear a beep
    * If you are already in column 1 on the screen (on any line) and you press `h`, you will hear a beep
  - You can move over multiple characters by preceding the movement keys with a number (any number of digits)
    * `3h` will move the cursor three characters to the left
    * `5j` will move the cursor down by five lines in the same column
  - Additional cursor movement commands
    * You can move the cursor on same screen using the following keys

Key	Effect
`^`	Move cursor to the first non-whitespace character on the line
`0`	Move cursor to first character on the line
`$`	Move cursor to the end of line
`H`	Move cursor to top left character on screen (High)
`L`	Move cursor to bottom left character on screen (Low)
`M`	Move cursor to the middle of screen
`w`	Move cursor forward by one word (delimited by punctuation)
`W`	Move cursor forward by one word (delimited by whitespace)
`b`	Move cursor backward by one word (delimited by punctuation)
`B`	Move cursor backward by one word (delimited by whitespace)
`e`	Move cursor to the last character of the word (or last character of next word, if already on the last character of current word), delimited by punctuation
`E`	Move cursor to the last character of the word (or last character of next word, if already on the last character of current word), delimited by whitespace
`(`	Move cursor to the beginning of sentence
`)`	Move cursor to the end of sentence
`{`	Move cursor to the beginning of paragraph
`}`	Move cursor to the end of paragraph
`G`	Go to last line of the document
n`G`	Go to nth line of the document, n is a number `3G` positions the cursor on the 3rd line

  - A sentence is delimited by a period followed by two spaces
  - A paragraph is delimited by an empty line (should not even contain whitespace)
  - These keys can also be preceded by a number to effect the movement
    - `4L` is a request to position cursor on the fourth line from the bottom (in column 1)
    - `6H` is a request to position cursor on the sixth line from the top (in column 1)

- Scrolling the screen
  - Scrolling is used to move the document on the screen up or down by a specified amount (just like Page Up and Page Down in word processors)
  - Use the following keys for scrolling (these are all control keys and `^F` implies hitting the key `F` while pressing down on the Control key)

Key	Effect
`^F`	Scroll forward one screen
`^B`	Scroll backward one screen
`^D`	Scroll down half screen
`^U`	Scroll up half screen
`^E`	Scroll forward one line
`^Y`	Scroll backward one line

  - As an aid to memory, notice that the key `Y` is next to `U` (for Up) and the key `E` is next to the key `D` (for Down)
  - All the keys in this case are case-insensitive

- Repositioning the screen
  - Use the key `z` followed by another key to reposition the screen while leaving the cursor in the current line of text

Key	Effect
z<CR>	Move current line to top of screen
z.	Move current line to middle of screen
z-	Move current line to bottom of screen

  `<CR>` is the key labeled Enter
- Getting information on current line
  - Press `^G` (ctrl-G) to get position information on current line
  - The output appears in the last line of the screen as follows

```
"unix05.tex" [Modified] line 189 of 192 --98%--
```

  - The same information can also be obtained by typing `:f`; the `:` commands belong to the underlying editor `ex`
  - Just the current line number is given by `:.`= followed by the Enter key
- Saving and quitting `vi`
  - Type `ZZ` (uppercase Z's) to save the file and quit
  - If you have modified the file, you will see a message like

```
"unix05.tex" 199 lines, 7776 characters
```

  - If you try to quit a file that has not been modified, it is not saved
    - If you just enter `vi` to create a new file and quit without making any changes, the new file is not created
- Searching for a pattern
  - Use `/` followed by the pattern
    - As you type the `/` in command mode, it appears at the last line of the screen
    - The pattern to be searched for appears in the last line immediately after the `/` as you type it
  - Search in the backward direction (towards top of file) is performed by changing the character `/` to `?`
    - You have to type `?pattern` to search for the pattern in backward direction
  - The search is repeated by using the following keys

Key	Effect
n	Repeat search in the same direction (forward or backward)
N	Repeat search in the opposite direction (forward or backward)
/<CR>	Repeat search in the forward direction
?<CR>	Repeat search in the backward direction

  where `<CR>` signifies the Enter key

- Deleting text
  - Use the command `d` to delete text
  - `d` is followed by one of the cursor movement keys to specify the amount of text to be deleted
    - `dd` deletes the entire line on which the cursor is placed
    - `5dd` will delete five lines starting with the current line
    - `dw` deletes to the end of word from the current character
    - `3dW` deletes three whitespace-delimited words from the current character
    - A single character is deleted by `x` or `dl`
    - The character before the cursor is deleted by `X` or `dh`
    - To delete from the current line to the bottom of screen, use `dL`
    - To delete from the current line to the end of document, use `dG`
    - To delete to the end of a paragraph, use `d}`
    - To delete until the occurrence of a certain pattern, use `d/`*pattern*`<CR>`, where *pattern* is the pattern to which you need to delete and `<CR>` is the Enter key
  - The delete commands are summarized as

Keys	Effect
`x`	Delete the character under cursor
n`x`	Delete n characters from current cursor position towards end of line
`X`	Delete the character just to the left of cursor
n`X`	Delete n characters to the left of cursor
`D`	Delete from cursor to the end of line
`d0`	Delete from the beginning of line up to the character left of cursor
`d$`	Delete from cursor to the end of line
`dd`	Delete current line
n`dd`	Delete n lines starting at the current line downwards
`d-`	Delete current line and the line just above it
n`d-`	Delete current line and n lines before it
`d+`	Delete current line and the line just below it
n`d+`	Delete current line and n lines after it
`dw`	Delete from cursor to the end of word
`dW`	Delete from cursor to whitespace towards right
`db`	Delete from cursor to beginning of current word
`dB`	Delete from cursor to whitespace towards left
`d/`*pattern*`<CR>`	Delete to the first occurrence of *pattern*
`d)`	Delete to end of sentence
`d}`	Delete to end of paragraph
`dL`	Delete to end of screen
`dG`	Delete to end of document

  - You can also use the `ex` delete commands
    - To delete from line a to line b, type

      `:`a`,`b`d<CR>`

      where a and b are two line numbers ($a < b$) and `<CR>` is the Enter key
    - To delete the first five lines (no matter where the cursor is situated), type `:1,5d<CR>`

- Cut and paste
  - The text that is deleted is automatically saved in an internal buffer
  - It can be retrieved (pasted) at any position by using the command `p` or `P`
  - `p` results into pasting the text immediately after the current cursor position while `P` pastes it immediately prior to cursor
  - Examples
    - Delete two lines by usng `2dd`
      Position the cursor where you want the lines to be pasted
      Press `p` to paste the lines
    - Swap two characters by typing `xp`
      - You are typing text and type `teh` when you meant to type `the`
        Position cursor on the character `e` in `teh`
        Press `xp` to cut and paste `e` after `h`
- Changing text
  - Use the command `c` to change text
  - This command uses the same delimiters from cursor movement commands as used by `d`
  - *Important: When you change text, you are put into Insert mode of* `vi` *and you have to press the Escape key to return to command mode; the only exception to the rule is the* `r` *command which accepts the new character and keeps you in command mode*
  - Examples
    - `cc` changes the entire line on which the cursor is placed
    - `5cc` will change five lines starting with the current line
    - `cw` changes to the end of word from the current character
    - `3cW` changes three whitespace-delimited words from the current character
    - A single character is changed by `r` (for replace) or `cl`
    - The character before the cursor is changed by `ch`
    - To change from the current line to the bottom of screen, use `cL`
    - To change from the current line to the end of document, use `cG`
    - To change to the end of a paragraph, use `c}`
    - To change until the occurrence of a certain pattern, use `c/`*pattern*`<CR>`, where *pattern* is the pattern to which you need to change and `<CR>` is the Enter key
    - To change text from cursor to end of line, press `C` or `c$`
  - An additional way to change text is provided by `s` (for substitute)
    - `s` allows you to type any number of characters in place of the current character (contrast with `r`)
    - `5s` allows to to type any number of characters over five characters starting with the character under cursor
    - You have to press Escape after you are done with substitution
  - `R` allows you to overwrite the line
    - The characters not overwritten are left as is

    - You have to press Escape to return to command mode
  - You can also change the case of letters under the cursor (from uppercase to lowercase, and vice versa) by pressing the tilde key (`~`)
    - The cursor moves one character to the right when you press `~`
- Joining lines
  - The current line and the line below it can be joined into a single line by `J` (uppercase)
  - n lines can be joined to the current line by typing n`J`
- Repeating the last command
  - The last command can be repeated by typing a period `.`
  - Position the cursor where you want the command to be repeated (e.g., to delete a line) and press `.`
  - Example
    - Delete a line using `dd`
    - Delete subsequent lines by pressing `.`
- Undoing the last command
  - The last command can be undone by hitting the key `u`
  - This allows you to undo the last command if you make an error, for example, delete a line that you did not intend to
  - The cursor could be anywhere in the document to undo the last command
  - The current line can be restored by typing `U` (uppercase)

Insert mode

- Insert mode allows you to add text into the file which is not possible in command mode
- You can enter insert mode from command mode by hitting the key `i`
  - `i` is not echoed to screen but whatever you type following `i` appears on the screen
- From insert mode, you can return to command mode by pressing the Escape key
  - *Always press the* Escape *key whenever you want to return to command mode, or whenever you want to verify that you are in command mode*
  - Pressing Escape moves the cursor one character to the left so that you are on the last character typed, and returns you to command mode
- Within insert mode, if you want to correct a typing error, press backspace key to erase the previously typed character
  - If the characters are still visible when you press backspace key, you can simply overwrite on them

  - The backspace key will not allow you to go past the point where you started the insert mode, or when a newline is started

- In general, `vi` does not tell you whether you are in insert mode or command mode
  - You can type the command

    ```
    :set showmode
    ```

    when you start `vi` in which case it will display that you are in insert mode at the bottom of the screen (towards right) when you are in insert mode; it does not show anything when you are in command mode

- `i` only allows you to insert before the current cursor position; if you want to insert *after* the current cursor position, you should use the key `a` (for append)
- All the commands that you use to enter insert mode (besides change text commands) are summarized below

Key	Effect
`i`	Insert at the current cursor position
`I`	Insert at the beginning of current line Beginning is defined as first non-whitespace character
`a`	Insert to the right of current cursor position
`A`	Append to the end of current line
`o`	Open a new line below the current line
`O`	Open a new line just above the current line

`ex` mode

- `ex` is the underlying line-based editor on which `vi` is based
  - `ex` has a full range of commands to do editing
  - We'll only cover the set of commands that are useful here
  - `ex` commands are useful in setting editing options and to perform global search and replace
- The `ex` mode is invoked by hitting the colon key (`:`) in command mode
  - A colon appears at the bottom left corner of the screen
  - The commands are echoed on the screen as you type them
  - The commands are not executed unless you hit the Enter key
  - The `ex` commands can be applied to a range of lines by specifying the first line and the last line (see examples below)
    - The current line is indicated by period (`.`)
    - The last line is indicated by `$`
    - Line number n is indicated by the integer `n`
    - The lines in entire file can be specified by `1,$` or the symbol `%`
- Saving and quitting a file

  - You can save the current editing by typing `:w` at any time
    * This will not throw you out of the `vi` session
  - You can save and quit by typing `:wq`
    * This is same as typing `ZZ` in `vi` command mode
  - If you have not made any changes to the file since the last save, you can also quit by typing `:q`
    * If you have made changes since the last save, and type `:q`, you get the following message
      ```
      No write since last change (use ! to override)
      ```
  - If you want to discard the changes made since last save, you can type `:q!` to quit `vi`
  - If you want to discard all the changes since the last save, but still want to stay in the editing session, type `:e!`
  - You can force `vi` to save the file by typing `:w!`
    * This may be required in some instances when `vi` complains about not being able to immediately save the file
  - You can save the file under a different name by typing `:w` followed by the new name for the file
  - Examples
    * Saving file under a different name
      ```
      :w filename
      ```
    * Saving lines 20 to 30 in `newfile`
      ```
      :20,30w newfile
      ```
    * Saving from current line to the end of file in `newfile`
      ```
      :.,$w newfile
      ```
    * Appending line number 1 through 20 to `newfile`
      ```
      :1,20w >> newfile
      ```

- Reading in a file
  - You can read in another file into the current `vi` session
  - Position the cursor on the line after which you have to read in the file named `foo`, and type the following
    ```
    :r foo
    ```
  - To read the file before the first line in the document, you can type
    ```
    :0r foo
    ```

- Search and replace
  - `vi` does not have its own command to perform the search and replace function but depends on the `ex` commands to do so
  - Search and replace is achieved by the `:s` command (for *substitution*)
  - The syntax is

    $$:m,ns/pat1/pat2/$$

    where *m* and *n* indicate the range of lines to which the command is applied

  - The command only replaces the first occurrence of *pat1* to *pat2*
  - To replace *all* occurrences, you should type a `g` (for global) after the last `/`
- Examples
  - Replace first occurrence of `teh` to `the` on current line
    `:s/teh/the/`
  - Replace all occurrences of `teh` to `the` on current line
    `:s/teh/the/g`
  - Replace all occurrences of `teh` to `the` on line numbers 15 through 50
    `:15,50s/teh/the/g`
  - Replace all occurrences of `teh` to `the` in entire document
    `:%s/teh/the/g`
  - Replace all occurrences of `teh` to `the` in entire document, but ask before doing so
    `:%s/teh/the/gc`
    There should be no space between `g` and `c`

Setting options in `vi`

- You can set the options in `vi` by using the `ex` mode
- Alternatively, you can create a file called `.exrc` in your home directory and set all the desired options as per your choice in there
- The syntax for setting the options is

`set option`	Enable `option`
`set nooption`	Disable `option`

- You can see all the available options and their current setting by typing the command

```
:set all
autoindent              nomodelines                     showmode
autoprint               nonumber                        noslowopen
autowrite               nonovice                        tabstop=8
beautify                optimize                        taglength=0
directory=/var/tmp      paragraphs=IPLPPPQPP LIpplpipnpbtags=tags /usr/lib/ta
noedcompatible          prompt                          tagstack
noerrorbells            noreadonly                      term=xterm
noexrc                  redraw                          noterse
flash                   remap                           timeout
hardtabs=8              report=5                        ttytype=xterm
noignorecase            scroll=11                       warn
nolisp                  sections=NHSHH HUuhsh+c         window=23
nolist                  shell=/usr/local/bin/tcsh       wrapscan
magic                   shiftwidth=8                    wrapmargin=1
mesg                    showmatch                       nowriteany
[Hit return to continue]
```

- My personal `.exrc` file sets the following options

```
set autoindent autowrite beautify noignorecase optimize
set shell=/usr/local/bin/tcsh showmatch
set showmode wrapscan wrapmargin=1
```

  - You do not need to type the colon in the `.exrc` file

- Different options are explained as under

autoindent Automatically indent code lines; useful to have when typing programs; default is `noautoindent`
abbreviation: `ai`, `noai`

autoprint Display changes after each editor command; For global replacement, display last replacement
abbreviation: `ap`, `noap`

autowrite Automatically save the file if changed before opening another file with `:n` or `:e`, or before giving a Unix shell command with `:!`; default is `noautowrite`
abbreviation: `aw`, `noaw`

beautify Ignore all control characters during input except tab, newline, and formfeed; default is `nobeautify`
abbreviation: `bf`, `nobf`

directory=/var/tmp Directory in which `ex` stores buffer files; buffer files are the temporary files that keep the changes made; specified directory must be user writeable
abbreviation: `dir`

errorbells Sound bell when an error occurs
abbreviation: `eb`, `noeb`

ignorecase Ignore case during a search; default is `noignorecase`
abbreviation: `ic`, `noic`

mesg Allow system to write messages to your screen when you are in editor

number Display line numbers in left column; default is `nonumber`
abbreviation: `nu`, `nonu`

showmatch Match the parenthesis and curly braces; useful for programming; default is `noshowmatch`
abbreviation: `sm`, `nosm`

showmode Display the insert mode at the bottom of the screen; default is `noshowmode`

warn Display the message "No write since last change."

wrapmargin=8 Establish a right margin of 8 characters on screen; if it is not zero, automatically inserts carriage return to break lines; default value is zero
abbreviation: `wm`

wrapscan Search wraps around the end of file
abbreviation: `ws`, `nows`

Executing Unix commands from within `vi`

- Unix commands can be executed from within `vi` in the `ex` mode using a shell escape

- To see who all are logged into the system, type the command

  `:!who`

- To apply the results of a command to a specified range of lines, just type the range between colon (`:`) and bang (`!`)
  - To show the examples in this document, I frequently type the command and follow it on next line with `:.!command`
  - An equivalent command to `:.!` is provided by `!!` (without the colon)
- To execute multiple shell commands from within `vi`, you can temporarily escape to shell by typing

  `:!`
  - To go to a shell of your choice, type in the shell name after `!`

    `:!tcsh`

Copying text

- You can use the `y` command to *yank* text into a temporary buffer, leaving the original text intact
- This command uses the same delimiters from cursor movement commands as used by `d`
- The copied text is pasted into the desired position by using `p` or `P` as described earlier
- Examples
  - `yy` or `Y` yanks the entire line on which the cursor is placed
  - `5yy` or `5Y` will yank five lines starting with the current line
  - `yw` yanks to the end of word from the current character
  - `3yW` yanks three whitespace-delimited words from the current character
  - A single character is yanked by `yl`
  - The character before the cursor is yanked by `yh`
  - To yank from the current line to the bottom of screen, use `yL`
  - To yank from the current line to the end of document, use `yG`
  - To yank to the end of a paragraph, use `y}`
  - To yank until the occurrence of a certain pattern, use `y/`*pattern*`<CR>`, where *pattern* is the pattern to which you need to yank and `<CR>` is the Enter key
  - To yank text from cursor to end of line, press `y$`

Using named buffers

- You can copy text into named buffers to be pasted at some other time during the same editing session
- Named buffers retain their contents when you move from one file to another within `vi`

- There are 36 named buffers available for users
  - Buffers named by a lowercase alphabetic character (from `a` to `z`) are explicitly assigned
    - An uppercase alphabetic letter can be used to append to the existing named buffer
  - Buffers named by a digit (from `0` to `9`) are implicitly assigned
- The named buffers are accessed by the double quotes character (`"`) in command mode for both copy as well as paste
- To copy a paragraph in buffer `a`, type

  ```
  "ay}
  ```

- You can then move to the appropriate location and paste the copied contents from buffer `a` by using

  ```
  "ap
  ```

- All the yank commands work as such in the named buffers

Editing multiple files

- You can edit more than one file in a single `vi` session
- Multiple files are not handled in `vi` simultaneously (for example, you cannot open two files in the same session but in different windows)
- You can go from one file to the other by using the command

  ```
  :e filename
  ```

- Before you go to edit a different file, you must save the current `vi` buffer by using the command `:w` or by using the following command in your `.exrc` file

  ```
  :set autowrite
  ```

- If you know the files to be edited in a single session, you can specify those on the command line to start `vi` as

  ```
  $ vi file1 file2 file3
  ```

  - `vi` starts editing with the first file (`file1`)
  - You can move to the next file by typing the command `:n`
  - You can move back to first file from within any file by typing the command `:rewind`
- If you want to discard the changes made to a file, you can always force exit to a next file by using the suffix `!` after `w`, `e`, or `n`
- As remarked earlier, named buffers retain their value and you can use those to copy and paste across files without having to use temporary files

Text markers

- Used to mark a location in the file
- They do not become part of the file text and are lost at the end of the editing session
- There are 26 marker names available in `vi`, corresponding to the lower case letters in the English alphabet (a to z)
- Markers are useful in large files to perform cursor movement, and to mark text for deletion or yanking
- Marker definitions are automatically lost if a line or character associated with the marker is deleted
- In `vi`, use the command `m` followed by an alphabetic key to create a text mark
- In `ex` mode, you can use `:k` followed by an alphebtic key to place the marker
- In both `vi` and `ex`, use the keys `'` or `` ` `` to return to the place marked
  - `'` returns you to the beginning of the line where you placed the marker
  - `` ` `` returns you to the exact location within the line where you placed the marker

Mapping Keys

- Key mapping is used to define macros and associate them with a specific key on the keyboard
- You can define the maps in the `.exrc` file
- You can also use the command `unmap` to cancel the map definition
- Example
  - Mapping the key `v` to swap two words

    ```
    map V dWElp
    ```
- You can see the current mappings of keys by typing the command `:map` in `vi`

Abbreviations

- Used in the input mode to type in long repetitive patterns
- Declared with the command `abbr`
- For example, the following command can be used in C programs to add long comments

  ```
  ab usa United States of America
  ```

  This will replace the three successive keystrokes of `usa` by the string `United States of America` without leaving the input mode
- You can see all the abbreviations by using the command `:abbr`

Starting vi at a specific location

- You can start the editing session at a specific line or the specific occurrence of a pattern
- The editing can be started at line 10 by typing

```
$ vi +10 filename
```

- You can start the editing session at a specific occurrence of a pattern by typing

```
$ vi +/pattern filename
```

Recovering files

- If an editing session is terminated due to any reason (power failure, system crash, modem disconnection), the edited file is preserved (you may lose the changes made on the last line)
- The file can be retrieved when you log in next
- The command to use is

```
$ vi -r filename
```

- Generally, you will have an email telling you that the file has been preserved and you can recover it with the command to recover it
- If you just type

```
$ vi -r
```

  it will inform you of the files that have been preserved for you

Editing and encrypting files

- The files can be saved as encrypted by using the `ex` mode command `:X`
- The encrypted files can be edited by using the `-x` option when starting `vi`
- *Be careful with the password assignment when you encrypt a file; if you forget the passowrd, you may not be able to recover the file*