Helix

A Post-Modern Text Editor

`$hx <filename>` open *<filename>* in Helix
`:q[uit]` Helix `:q!` force-quit (no save)
`:tutor` open the tutorial

Unlike Vim, in Helix you first select what you wish to operate on, and then perform the operation.

[mode] [count] [select] [operation]
optional *optional*

So, to delete three words in Helix, it's `v3wd`

- `v` - enter select(extend) mode
- `3` - number of items to act on
- `w` - which would be 'words'
- `d` - perform the delete operation

␣? open command palette

*n*gg or *n*G jump to line *n*
:*n* will do this too

gt go to top of screen · gk move up (textual) · gm go to last modified file · ga go to last accessed file · g. go to last modification · gc go to middle of screen · gj move down (textual) · gb go to bottom of screen

first line **gg** · move page up **^b** · move ½ page up **^u** · move up **k** · move down **j** · move ½ page down **^d** · move page down **^f** · last line **ge**

go to 1st char **gs** · goto line start **gh** · move previous WORD start **B** · move previous word start **b** · move left **h** · *n*<op> do *n* times · move right **l** · move next word start **w** · move next word end **e** · move next WORD start **W** · move next WORD end **E** · goto line end **gl**

9 words
`https://www.helix-editor.com`
1 WORD

Editing

~	switch **case** of selected text
`	to **lower** case
⌥`	to **upper** case
.	repeat **last insert**
^a	**increment** object under cursor
^x	**decrement** object under cursor
_	trim **whitespace** from selection
:sort	**sort** ranges in selection
:rsort	**reverse-sort** ranges in selection
:ear	jump to **earlier** point in edit history
:lat	jump to **later** point in edit history
:char	get **info** about char under primary cursor
:diffg	reset the **diff change** at the cursor position
:encoding	set encoding

Based on https://encoding.spec.whatwg.org

Modes

- normal mode
- v select (extend) mode*
- g goto mode
- m match mode *surround*
- i insert mode
- : command mode
- z view mode
- Z sticky view mode
- ^w window mode ␣w
- ␣ space mode

** movement extends selection, rather than replacing it*

Files

Force ! *ignores unsaved changes* — optional path *

␣f	open file **picker** (␣F open file picker at **cwd**)
:o*pen*	**open** a file
:n*ew*	create a **new** scratch buffer
:w*rite*	**write** buffer to disk `:w some/path.txt` `:w!` *
:u*pdate*	write buffer **only if modified**
:wa	write **all** buffers to disk
:wq *or* :x	write buffer to disk and **close** *current view* `:wq!` *or* `:x!` *
:wbc	write buffer to disk and **close** *buffer* `:wbc!` *
:wqa *or* :xa	write **all** buffers to disk and close all views `:wqa!` *or* `:xa!`
:reload	discard changes and **reload** from source
:reload-all	discard changes and reload all docs from source

Buffers

␣b	open buffer **picker**
gn	go to **next** buffer *alt* :bn
gp	go to **previous** buffer *alt* :bp
:bc	**close** current buffer
:bco	close all **other** buffers
:bca	close **all** buffers without quitting
:bc!	close current buffer
␣Y	close all other buffers
␣R	close all buffers without quitting

"save as" `:w some/path.txt`

View

use Z (sticky view mode) to repeat actions

␣wo close all **other** windows · ␣wq close current window

z^b move page **up** · z^u move ½ page **up**

zk scroll the view **upwards**

swap window **upwards** ␣wK

swap window **downwards** ␣wJ

:hs open file in **horizontal split** · :hnew new scratch buffer in horizontal split

swap window to **left** ␣wH

swap window to **right** ␣wL

align to **top** of screen zt

:vs open file in **vertical split** · :vnew new scratch buffer in vertical split

vertically **centre** the line zz

scroll the view **downwards** zj

move page **down** z^f · move ½ page **down** z^d

align to **bottom** of screen zb

move to split above ␣wk · move to left split ␣wh · move to right split ␣wl · move to split below ␣wj

add this to `config.toml`:

```
[keys.normal]
C-j = "jump_view_down"
C-k = "jump_view_up"
C-h = "jump_view_left"
C-l = "jump_view_right"
```

Shell

- | pipe each selection though shell command, replacing with output
- ⌥| pipe each selection though shell command, ignoring output
- ! run shell cmd, inserting output before each selection
- ⌥! run shell cmd, append output after each selection
- $ pipe each selection to the shell command, keep selections where command returned 0
- :cd change cwd
- :pwd print working directory
- :pipe pipe each selection to the shell command
- :pipe-to pipe each selection to the shell command, ignoring output
- :sh run shell command

Inserting Text

start of line **I** · before cursor **i** · after cursor (append) **a** · end of line **A**

previous line **O** · next line **o** · replace character **r** · replace with yanked text **R**

undo commit checkpoint **^s** · auto-complete **^x** · insert register content **^r**

- ␣f open **file** picker
- ␣b open **buffer** picker
- ␣j open **jumplist** picker
- ␣? open **command** palette
- ␣' open **last fuzzy** picker

c **change** selection (delete & enter insert mode) ⌥c (without yanking)

d **delete** selection ⌥d (without yanking)

x select **current line** *n*x *n* lines

% select **entire file**

; **collapse selection** to single cursor ⌥; flip selection cursor & anchor

, keep **only primary** selection ⌥, remove primary selection

u undo · ⌥u move **backward** in **history**

U redo · ⌥U move **forward** in **history**

^c toggle **comment**

:fmt **format** using LSP

> indent

< unindent

Pickers

^t	toggle **preview**
↑ *or* ^p	**previous** entry
↓ *or* ^n	**next** entry
Home	go to **first** entry
End	go to **last** entry
PageUp *or* ^u	page up
PageDown *or* ^d	page up
Enter	**open** selected
^s	open **horizontally**
^v	open **vertically**
⎋ *or* ^c	**close** picker

⎋ escape · ⌥ option (Alt-) · ^ control · ⌘ command · ␣ space

In Helix: there are no named bookmarks, but you can save a location in the jumplist with ^s, then jump back to that location by opening the jumplist picker with ␣j, or back in the jumplist with ^o and forward with ^i.

Fx **find** next char — selects up to and including *x* — ← Backward

fx Forward →

Tx **find** 'til next char — selects up to *x*

tx

f, F, t and T are not confined to the current line (unlike Vim)

⌥. repeat last motion *f, t or m*

t<ret> selects 'til the newline represented by <ret>

Search

- / **search** for regex pattern
- ? search for previous pattern
- n select **next** search match
- N select **previous** search match
- * use current selection as search pattern

Jumplist

- ^s **save** current selection to jumplist
- ␣j open jumplist **picker**
- ^i jump **forward** on jumplist
- ^o jump **backward** on jumplist

Selection Manipulation

s	select all **regex** matches inside selections
S	**split** selection in to sub-sections on **regex** matches
⌥s	split selection on **newlines**
⌥_	**merge** consecutive selections
&	**align** selection in columns
⌥;	**flip** selection cursor and anchor
⌥:	**ensure** selection is in **forward** direction
⌥,	**remove** primary selection
C	**copy** selection onto **next** line (add cursor below)
⌥C	copy selection onto previous ine (add cursor above)
(	**rotate** main selection **backward**
)	**rotate** main selection **forward**
⌥(	**rotate** selection **contents backward**
⌥)	**rotate** selection **contents forward**
X	**extend** selection to line bounds (line-wise selection)
⌥x	**shrink** selection to line bounds (line-wise selection)
J	**join** lines inside selection
⌥J	join lines inside selection and select inserted space
K	**keep** selections matching the regex
⌥K	**remove** selections matching the regex
⌥o	**expand** selection to parent syntax node (TS)
⌥i	**shrink** syntax tree object selection (TS)
⌥p	select **pevious sibling** node in syntax tree (TS)
⌥n	select **next sibling** node in syntax tree (TS)

Copying & Pasting

y	**yank** selection
p	**paste after** selection
P	**paste before** selection
"<reg>	select **register** to yank or paste from
␣p	paste system clipboard **after** selections
␣P	paste system clipboard **before** selections
␣y	join and yank selections to clipboard
␣Y	yank main selection to clipboard
␣R	replace selections by clipboard contents

Registers

"ay yank the current selection to register 'a'
"ap paste the text in register 'a' after the selection
"ha store the selection in register 'a' then **change** it
"ad store the selection in register 'a' then **delete** it

Special Registers:

- / last search
- : last executed command
- " last yanked text
- _ 'black-hole'; no-op. *No data will be read or written to the _ register.*

:clear-register **clear** given register

Go To

unimpaired

←Previous		Next→
[d	diagnostic	]d •
[f	function	]f •
[t	type definition	]t •
[a	argument/ parameter	]a •
[c	comment	]c •
[T	test	]T •
[p	paragraph	]p
[g	change	]g
[␣	newline	]␣

[D ←first · last→]D · [G ←first · last→]G · [␣ ↑above · below↓]␣

• requires an active language server for the file
• requires a tree-sitter grammar for the file type

LSP

gd	go to **definition**
gy	go to **type definition**
gr	go to **references**
gi	go to **implementation**
␣k	show **documentation** for item under cursor
␣s	open document **symbol** picker
␣S	open **workspace** symbol picker
␣d	open document **diagnostics** picker
␣D	open **workspace** diagnostics picker
␣r	**rename** symbol
␣a	apply **code action**
␣h	select symbol **references**

:lsp-restart
:lsp-stop

Surround

mm	goto **matching bracket** (TS)
ms*<char>*	surround **current selection** with *<char>*
mr*<from><to>*	replace surround **character** *<from>* with *<to>*
md*<char>*	**delete** surround character *<char>*
ma*<object>*	select around textobject
mi*<object>*	select inside textobject

key after mi or ma

w word	W WORD
P paragraph	c class
f function	a argument/parameter
m closest surround pair	o comment
(, [, ', *etc* specify surround pairs	t test
	g change

Config

- :theme — change editor theme
- :get — get the value of a config option
- :set — set the value of a config option
- :toggle — toggle a config option
- :lang — set buffer language
- :config-reload — refresh config
- :config-open — open config.toml
- :config-open-workspace — open workspace config.toml

Search for the word under the cursor

⌥o*n (LSP) or be*n

If there's an LSP, ⌥o expands selection to the parent syntax node (with would be the word in this case). Then * uses the current selection as the search pattern, and n goes to the next occurence. b selects to the begining of the word, and e selects to the end of the word, effectively selecting the whole word.

Global Replace (for replace inside selection omit %)

%s*regex*<ret>c*replacement*<esc>

% selects the entire buffer, s opens a prompt for a regex, <ret> validates the regex and replaces the selection with one per match (hence, all occurences of *regex* are selected). c deletes the selection contents and enters insert mode, *replacement* is typed and then <esc> goes back to normal mode.

Block Selection

There's no "block selection" mode, so instead you'd use multiple cursors. Expand your block selection vertically by adding new cursors on the line below with C, and horizontally using standard movements.

Helix Cheat Sheet v1.1
Steve Hoy 2023-09-25
Helix 23.05 (7f5940be)