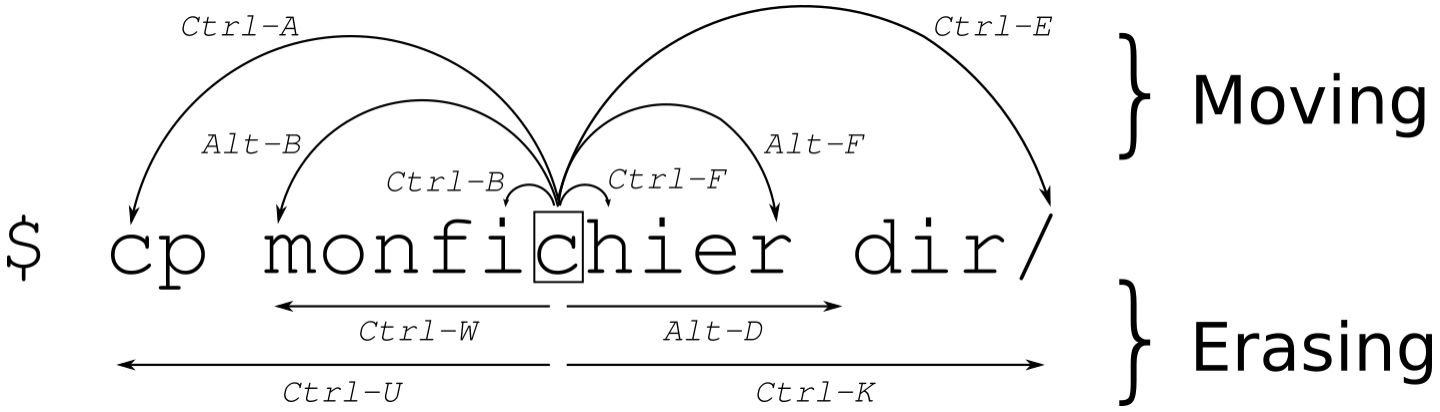
Useful bash / zsh shortcuts

MacOS iTerm 2 users must turn on meta key — https://coderwall.com/p/_lmivq

Nice visual cheatsheet from the article:



Move cursor

Ctrl + a	Go to the beginning of the line (Home)
Ctrl + e	Go to the End of the line (End)
Alt + b	Back (left) one word
Alt + f	Forward (right) one word
Ctrl + f	Forward one character
Ctrl + b	Backward one character
Ctrl + xx	Toggle between the start of line and current cursor position

Edit

Ctrl + u	Cut the line before the cursor position
Alt + Del	Delete the Word before the cursor
Alt + d	Delete the Word after the cursor
Ctrl + d	Delete character under the cursor
Ctrl + h	Delete character before the cursor (backspace)
Ctrl + w	Cut the Word before the cursor to the clipboard
Ctrl + k	Cut the Line after the cursor to the clipboard
Alt + t	Swap current word with previous
Ctrl + t	Swap the last two characters before the cursor (typo)
Esc + t	Swap the last two words before the cursor.
Ctrl + y	Paste the last thing to be cut (yank)
Alt + u	UPPER capitalize every character from the cursor to the end of the current word.
Alt + l	Lower the case of every character from the cursor to the end of the current word.
Alt + c	Capitalize the character under the cursor and move to the end of the word.
Alt + r	Cancel the changes and put back the line as it was in the history (revert)
Ctrl + _	Undo

History

Ctrl + r	Recall the last command including the specified character(s)(equivalent to : vim ~/.bash_history).
Ctrl + p	Previous command in history (i.e. walk back through the command history)
Ctrl + n	Next command in history (i.e. walk forward through the command history)
Ctrl + s	Go back to the next most recent command.
Ctrl + o	Execute the command found via Ctrl+r or Ctrl+s
Ctrl + g	Escape from history searching mode
Alt + .	Use the last word of the previous command

Process control

Bang(!) - The History Expansion

Bash also has some handy features that use the ! (bang) to allow you to do some funky stuff with bash commands.
General notation is `'![event][:word[:modifier[:modifier]...]]'`.
You may ommit word separator `':'`, if the word designator begins with a `'^'`, `'$'`, `'*'`, `'-'`, or `'%'`.
If a word designator is supplied without an event specification, the previous command is used as the event.
After the optional word designator, you can add a sequence of one or more modifiers, each preceded by a `':'`.

Events	Meaning	Example
!	Start a history substitution, except when followed by a space, tab, the end of the line, '=' or '(' (when the extglob shell option is enabled using the shopt builtin).	

!*n*	Refer to command line ***n***.	`$ history`
		`1 echo foo bar baz`
		`2 history`
		`$ !1`
		`#Print command that will be saved in history`
		`#+and executed`
		`echo foo bar baz`
		`#Actual execution`
		`foo bar baz`

!-*n*	Refer to the command ***n*** lines back.	`$ history`
		`1 echo foo`
		`2 echo bar`
		`3 echo baz`
		`4 history`
		`$ !-3`
		`echo bar`
		`bar`

!!	Refer to the previous command. This is a synonym for '!-1'.	`$ echo foo bar baz`
		`foo bar baz`
		`$ !!`
		`echo foo bar baz`
		`foo bar baz`

!*string*	Refer to the most recent command preceding the current position in the history list starting with ***string***.	`$printf '%s\n' foo`
		`foo`
		`$ echo bar`
		`bar`
		`$ !pri`
		`printf '%s\n' foo`
		`foo`

!?*string*[?]	Refer to the most recent command preceding the current position in the history list containing ***string***. The trailing '?' may be omitted if the ***string*** is followed immediately by a newline.	`$printf '%s\n' foo`
		`foo`
		`$ echo bar`
		`bar`
		`$ !?ntf`
		`printf '%s\n' foo`
		`foo`
		`$ !?bar`
		`echo bar`
		`bar`

^*string1*^*tring2*^	Quick Substitution. Repeat the last command, replacing ***string1*** with ***string2***. Equivalent to `!!:s/string1/string2`.	For more info, refer to `s/old/new/` in **Modifiers** section.
		`$ echo foo`
		`foo`
		`$ ^echo^printf '%s\n'^`
		`printf '%s\n' foo`
		`foo`

!#	Repeat entire command line before this event.	`$ echo foo; echo bar; !#echo baz`
		`echo foo; echo bar; echo foo; echo bar; echo baz`
		`foo`
		`bar`
		`foo`
		`bar`
		`baz`

Words	**Meaning**	**Example**

0 (zero)	The 0th word. For many applications, this is the command word.	`$ echo foo`
		`foo`
		`$ !:0`
		`echo`

n	The ***n***th word.	`$ echo foo bar baz`
		`foo bar baz`
		`$ echo !:2`
		`echo bar`
		`bar`

^	The first argument; that is, word 1.	`$ echo foo bar baz`
		`foo bar baz`
		`$ echo !^`
		`echo foo`
		`foo`

`$`	The last argument.	`$ echo foo bar baz`
		`foo bar baz`
		`$ echo !$`
		`echo baz`
		`baz`

%	The word matched by the most recent `?string?` search	`$ echo foo`
		`foo`
		`$ printf '%s\n' bar`
		`bar`
		`$ !?ch`
		`echo foo`
		`foo`
		`$ !% baz`
		`echo baz`
		`baz`
		`$ !?bar`
		`printf '%s\n' bar`
		`bar`
		`$ echo !%`
		`echo bar`
		`bar`

x-y	A range of words; `-y` abbreviates `0-y`.	`$ echo foo bar baz`
		`foo bar baz`
		`$ echo !:2-3`
		`echo bar baz`
		`bar baz`
		`$ !:-1`
		`echo bar`
		`bar`

*	All of the words, except the 0th. This is a synonym for `1-$`. It is not an error to use `*` if there is just one word in the event - the empty string is returned in that case.	`$ echo foo bar baz`
		`foo bar baz`
		`$ printf '%s\n' !*`
		`printf '%s\n' foo bar baz`
		`foo`
		`bar`
		`baz`

x*	Abbreviates `x-$`	`$ echo foo bar baz`
		`foo bar baz`
		`$ printf '%s\n' !:2*`
		`printf '%s\n' bar baz`
		`bar`
		`baz`

x-	Abbreviates `x-$` like `x*`, but omits the last word.	`$ echo foo bar baz`
		`foo bar baz`
		`$ printf '%s\n' !:0-`
		`printf '%s\n' echo foo bar`
		`echo`
		`foo`
		`bar`

Modifiers	Meaning	Example

p	Print the new command but do not execute it. Printed command is saved in history, so you can use `Ctrl+p` to re-enter it in current prompt.	`$ echo foo bar baz`
		`foo bar baz`
		`$ !:p`
		`#Printed, but not executed`
		`echo foo bar baz`
		`$ !:*:p`
		`foo bar baz`

h	Remove a trailing pathname component, leaving only the head (Actually, remove all after last `/`, including).	`$ echo foo /example/path/bar.txt baz`
		`foo /example/path/bar.txt baz`
		`$ !:p:h`
		`echo foo /example/path`

t	Remove all leading pathname components, leaving the tail (Actually, remove all before last `/`, including).	`$ echo foo /example/path/bar.txt baz`
		`foo /example/path/bar.txt baz`
		`$ !:p:t`
		`bar.txt baz`

r	Remove a trailing suffix of the form `.suffix`, leaving the basename (Actually, remove	`$ echo foo /example/path/bar.txt baz`
		`foo /example/path/bar.txt baz`
		`$ !:p:r`
		`echo foo /example/path/bar`

	all after last `.`, including).	

e	Remove all but the trailing suffix (Actually, remove all before last `.`, including).	`$ echo foo /example/path/bar.txt baz`
		`foo /example/path/bar.txt baz`
		`$ !:p:e`
		`txt baz`

q	Quote the substituted words, escaping further substitutions.	`$ echo foo 'bar baz'`
		`foo bar baz`
		`$ !:p:q`
		`'echo foo '\''bar baz'\'''`

x	Quote the substituted words as with ‘q’, but break into words at spaces, tabs, and newlines.	`$ echo foo 'bar baz'`
		`foo bar baz`
		`$ !:p:x`
		`'echo' 'foo' ''\''bar' 'baz'\'''`

s/old/new/	Substitute ***new*** for the first occurrence of ***old*** in the event line. Any delimiter may be used in place of `/`. The delimiter may be quoted in ***old*** and ***new*** with a single backslash. If `&` appears in ***new***, it is replaced by ***old***. A single backslash will quote the `&`. The final delimiter is optional if it is the last character on the input line.	`$ echo foo bar`
		`foo bar`
		`$ !:p:s/foo/baz`
		`echo baz bar`

&	Repeat the previous substitution.	`$ echo foo bar`
		`foo bar`
		`$ !:p:s/foo/baz`
		`echo baz bar`
		`$ printf '%s\n' foo`
		`foo`
		`$ !:p:&`
		`printf '%s\n' baz`

g
a
	Cause changes to be applied over the entire event line. Used in conjunction with `s`, as in gs/old/new/, or with `&`.	`$ echo foo bar foo`
		`foo bar foo`
		`$ !:p:gs/foo/baz`
		`echo baz bar baz`

G	Apply the following ‘s’ modifier once to each word in the event.	Result is same as in `g` modifier

Recent links

- Bash Shortcuts For Maximum Productivity
- Syntax Bashkeyboard
- Moving efficiently in the CLI
- Bash History Expansion