Bash Binds

The completion keybinds use mnemonic symbols for each category. `@` for hostname, `/` pathsep for files, `!` bang for commands. Known hosts won't complete if they are hashed, which they probably should be. Most of Bash syntax has similar mnemonics in the syntax for keys & string expansions, but it sounds a bit weird explaining it.

Emacs Bindings

```
abort C-g, C-xC-g, M-C-g
accept-line C-j, C-m

backward
├── char C-b, <left>
├── delete C-h
├── kill
│   ├── line C-x C-?
│   └── word M-C-h
└── word M-b, M-<left>

beginning-of
├── history M-<, <pgup>
└── line C-a, <home>

call-last-kbd-macro C-x e
capitalize-word M-c

character-search C-]
└── backward M-C-] $char

clear
├── display M-C-l
└── screen C-l

complete C-i, <esc> <esc>
├── command M-!
├── filename M-
├── hostname M-@
├── into-braces M-{
├── username M-~
└── variable M-$

delete
├── char C-d, <delete>
└── horizontal-space M-\
```

```
digit
├── argument M--
├── argument M-0
├── argument M-1
├── argument M-2
├── argument M-3
├── argument M-4
├── argument M-5
├── argument M-6
├── argument M-7
├── argument M-8
└── argument M-9

display-shell-version C-x C-v
downcase-word M-l
dynamic-complete-history M-C-i
edit-and-execute-command C-x C-e

end
├── kbd-macro C-x )
└── of-history M->, history M-<pgdown>
    └── line C-e, line <end>

exchange-point-and-mark C-x C-x
execute-named-command M-x

forward
├── char C-f, M-<right>
├── search-history C-s
├── word M-f
└── word M-<right>

glob
├── complete-word M-g
├── expand-word C-x*
└── list-expansions C-xg

history-expand-line M-^
```

```
insert
├── comment M-#
├── completions M-*
└── last-argument M-., M-_

kill
├── line C-k
└── word C-<delete>, M-d

next
└── history C-n, M-<down>

non-incremental
├── forward-search-history M-n
└── reverse-search-history M-p

operate-and-get-next C-o

possible
├── command-completions C-x!
├── completions M-, M-?
├── filename-completions C-x *
├── hostname-completions C-x@
├── username-completions C-x~
└── variable-completions C-x$

previous
└── history C-p, M-<up>

quoted-insert C-q, C-v, <insert>
re-read-init-file C-x C-r
reverse-search-history C-r
revert-line M-C-r, M-r
set-mark C-@, M-"

shell
├── backward-word M-C-b
├── expand-line M-C-e
├── forward-word M-C-f
├── kill-word M-C-d
└── transpose-words M-C-t
```

```
spell-correct-word C-x s
start-kbd-macro C-x (
tilde-expand M-&

transpose
├── chars C-t
└── words M-t

undo C-_, C-x C-u

unix
├── line-discard C-u
└── word-rubout C-w

upcase-word M-u

yank C-y
├── last-arg M-., M-_
├── nth-arg M-C-y
└── pop M-y
```

Self-Insert

Hitting `C v` or `<insert>` runs `quoted-insert $key` which inserts the escape sequence for `$key` into the terminal. Running `quoted-insert` on these keys in sequence enters the following into the terminal. Successive `C v` keybinds seem to be handled recursively.

	000	001	010	011	100	101	110	111
		S	M	M S	C	C S	C M	C M S
<insert>	^[[2~		^[[2;3~	^[[2;4~	^[[2;5~	^[[2;6~	^[[2;7~	^[[2;8~
C v	^v		^[V					

The terminal application will intercept `C v` before it's given to the window manager (something like this), so you need to use `C S v` to paste.

In `xkb`, the `shift` modkey causes alphanumeric keybinds to be interpreted as their raw character codes. `C S v` literally inserts `C V` and `AltGr ;` into `¶`. Thus, I removed `self-insert` & `do-lowercase-version` binds from this reference.

Escape Sequences

F1	^[OP	F5	^[[15~	F9	^[[20~	home	^[[H	del	^[[3~
F2	^[OQ	F6	^[[17~	F10	^[[21~	end	^[[F	bksp	^?
F3	^[OR	F7	^[[18~	F11	^[[23~	pgup	^[[5~	bktab	^[[Z
F4	^[OS	F8	^[[19~	F12	^[[24~	pgdn	^[[6~	esc	^[

Quality of Life Keybinds

```
abort C-g
capitalize-word M-c

character-search C ] $char
└── backward M C ] $char

complete
├── command M !
├── filename M \*
├── hostname M @
├── into-braces M {
├── username M ~
└── variable M $
```

```
digit-argument M [0-9-]
downcase-word M l
dynamic-complete-history M C i
edit-and-execute-command C xC e
exchange-point-and-mark C xC x

glob
├── complete-word M g
├── expand-word C x
└── list-expansions C xg

insert
├── completions <esc> *
└── last-argument M ., M _
```

```
print-last-kbd-macro
quoted-insert C-v, <insert>
undo C _
undo C xC u
upcase-word M-u

yank
├── nth-arg M C y
└── pop M y
```

Keybind Combos

To avoid the strange `unix-word-rubout`

Keybinds involving `C-w` require setting:

```
bind '"\C-@":set-mark'
bind '"\e ":set-mark'
bind '"\C-w":kill-region'
bind '"\ew":copy-region-as-kill'
```

From middle of file or url path `C-@ M-C ] <space> C-x C-x C-w`

Jump to last space, cut region, leaving the remainder of the path.
Then `C-a varname=value; C-x C-x <right>`

From midline `C-@ C-] | C-] | C-x C-x C-w`

Set mark with `C-@`, search forward the next two `|` chars, then `exchange-point-and-mark` with `C-x C-x`.
This highlights the region. Then `C-w` to cut the text with `kill-region`.

From midline `=C-k C-a C-y`

Swap order of commands

From midline `C-k C-a C-] | C-] | ... C-y`

cut to end of line, search through line for `|` char and insert the tail of a bash pipeline. Simply repeating `C-]` doesn't work.

From midline `C-a export VAR=value;`

Prefix command with environment var

From midline `C-k # C-y`

Kill to end of line, insert `#` and paste. `C-y` to `yank`. Commented command content is still in history.

Running `print-last-keyboard-macro ...`

Shows you what's recorded. You can make composite functions.

Unmapped by default

```
alias-expand-line
arrow-key-prefix

backward
└── byte

bash-vi-complete

copy
├── backward-word
├── forward-word
└── region-as-kill

dabbrev-expand

delete
└── char-or-list
```

```
dump
├── functions
├── macros
└── variables

emacs-editing-mode
export-completions
fetch-history

forward
├── backward-delete-char
└── byte

history
├── and-alias-expand-line
├── search-backward
├── search-forward
└── substring
    ├── search-backward
    └── search-forward
```

```
kill
├── region
└── whole-line

magic-space

menu
├── complete
└── complete-backward

next
└── screen-line

non-incremental
├── forward-search-history-again
└── reverse-search-history-again

old-menu-complete
overwrite-mode
```

```
previous
└── screen-line

print-last-kbd-macro
redraw-current-line

shell
└── backward-kill-word

skip-csi-sequence
tab-insert
tty-status
universal-argument

unix
└── filename-rubout
```

Vi Bindings

For vi bindings ... you're on your own.

```
vi
├── append
│   ├── eol
│   └── mode
├── arg-digit
├── back-to-indent
├── backward
│   ├── bigword
│   └── word
├── bword
├── bWord
└── change
    ├── case
    ├── char
    └── to
```

```
vi
├── char-search
├── column
├── complete
├── delete-to
├── edit-and-execute-command
├── editing-mode
├── end
│   ├── bigword
│   └── word
├── eof-maybe
├── eword
├── eWord
├── fetch-history
└── first-print
```

```
vi
├── forward
│   ├── bigword
│   └── word
├── fword
├── fWord
├── goto-mark
├── insert-beg
├── insertion-mode
├── match
├── movement-mode
├── next-word
├── overstrike-delete
├── prev-word
└── put
```

```
vi
├── redo
├── replace
├── rubout
├── search-again
├── set-mark
├── subst
├── tilde-expand
├── undo
├── unix-word-rubout
└── yank
    ├── arg
    ├── pop
    └── to
```