

Advanced Programming in the UNIX Environment

Week 05, Segment 2: Unix Development Tools: The Editor

**Department of Computer Science
Stevens Institute of Technology**

Jan Schaumann

`jschauma@stevens.edu`


`https://stevens.netmeister.org/631/`

Software Development Tools

The UNIX Userland is an IDE – essential tools that follow the paradigm of “Do one thing, and do it right” can be combined.

You are here.

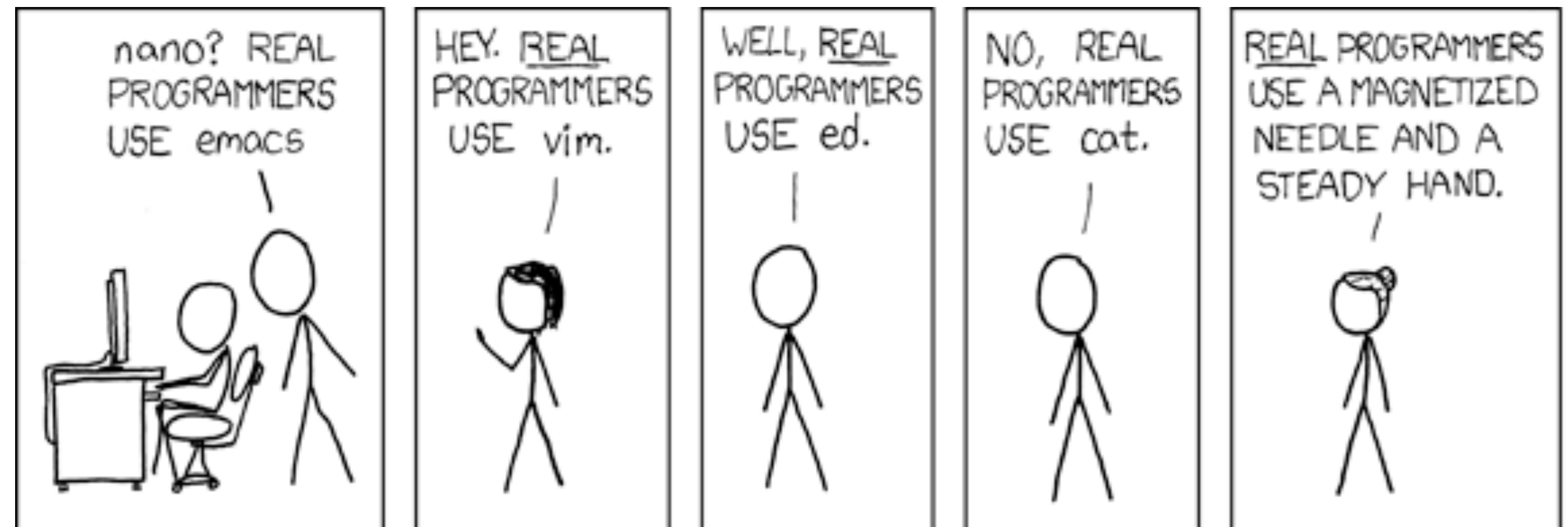
The most important tools are:

- `$EDITOR` 
- the compiler toolchain
- `gdb(1)` – debugging your code
- `make(1)` – project build management, maintain program dependencies
- `diff(1)` and `patch(1)` – report and apply differences between files
- `cvs(1)`, `svn(1)`, `git(1)` etc. – revision control, distributed project management

\$EDITOR

Know your \$EDITOR. Core functionality:

- syntax highlighting
- efficient keyboard maneuvering
- setting markers, using buffers
- copy, yank, fold e.g. blocks
- search and replace
- window splitting
- autocompletion
- jump to definition / manual page
- applying external commands and filters



Partial <https://xkcd.com/378/>

CS631 - Advanced Programming in the UNIX Environment

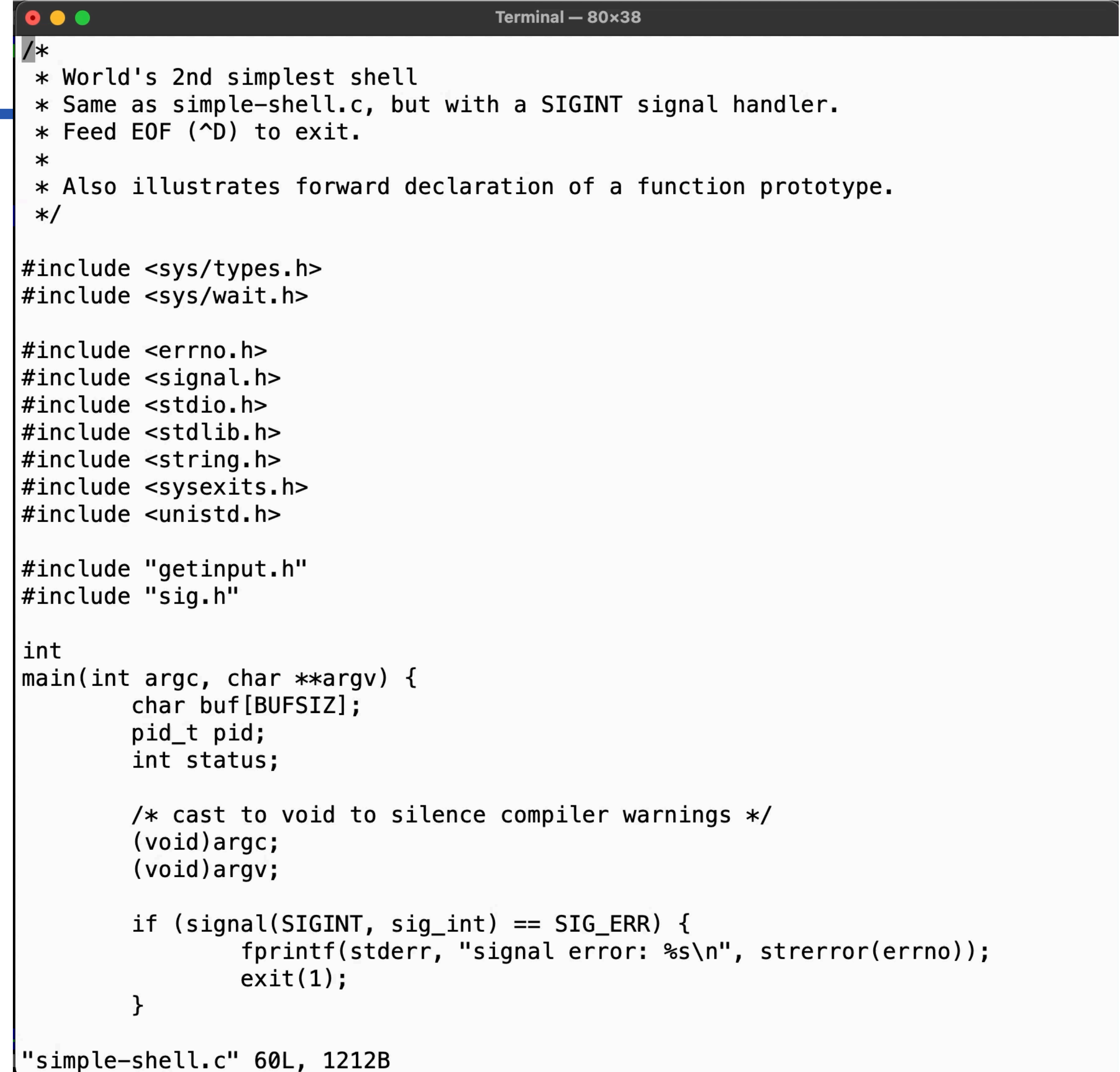


<https://dave.cheney.net/2017/08/21/the-here-is-key>

\$EDITOR Examples: vim

Efficient keyboard maneuvering:

- up, down, left, right (h, j, k, l)
- move by word, go to end (w, b, e)
- search forward, backward, move to beginning or end of line (/ , ? , ^ , \$)
- page up or down (^D, ^B)
- center page, top or bottom (zz, zt, zb)
- move to matching brace, move to beginning/end of code block (%,] }, [{)
- move through multiple files (:n, :prev, :rew)

A terminal window titled "Terminal — 80x38" displays the source code of a file named "simple-shell.c". The code is a C program for a simple shell. It includes headers for system types, wait, errno, signal, stdio, stdlib, string, sysexits, and unistd. It defines a main function that takes argc and argv, declares a buffer, pid_t, and int status. It casts argc and argv to void to silence compiler warnings. It sets a SIGINT signal handler to sig_int. If the signal handler returns SIG_ERR, it prints an error message to stderr and exits with status 1. The terminal window has a dark background and a light-colored text area.

```
/*
 * World's 2nd simplest shell
 * Same as simple-shell.c, but with a SIGINT signal handler.
 * Feed EOF (^D) to exit.
 *
 * Also illustrates forward declaration of a function prototype.
 */

#include <sys/types.h>
#include <sys/wait.h>

#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sysexits.h>
#include <unistd.h>

#include "getinput.h"
#include "sig.h"

int
main(int argc, char **argv) {
    char buf[BUFSIZ];
    pid_t pid;
    int status;

    /* cast to void to silence compiler warnings */
    (void)argc;
    (void)argv;

    if (signal(SIGINT, sig_int) == SIG_ERR) {
        fprintf(stderr, "signal error: %s\n", strerror(errno));
        exit(1);
    }
}
```

"simple-shell.c" 60L, 1212B

\$EDITOR Examples: vim

Efficient keyboard maneuvering:

- up, down, left, right (h, j, k, l)
- move by word, go to end (w, b, e)
- search forward, backward, move to beginning or end of line (/ , ? , ^ , \$)
- page up or down (^D, ^B)
- center page, top or bottom (zz, zt, zb)
- move to matching brace, move to beginning/end of code block (%,] }, [{)
- move through multiple files (:n, :prev, :rew)

```

Terminal — 80x38
#include "sig.h"

int
main(int argc, char **argv) {
    char buf[BUFSIZ];
    pid_t pid;
    int status;

    /* cast to void to silence compiler warnings */
    (void)argc;
    (void)argv;

    if (signal(SIGINT, sig_int) == SIG_ERR) {
        fprintf(stderr, "signal error: %s\n", strerror(errno));
        exit(1);
    }

    while (getinput(buf, sizeof(buf))) {
        buf[strlen(buf) - 1] = '\0';

        if((pid=fork()) == -1) {
            fprintf(stderr, "shell: can't fork: %s\n",
                strerror(errno));
            continue;
        } else if (pid == 0) { /* child */
            execlp(buf, buf, (char *)0);
            fprintf(stderr, "shell: couldn't exec %s: %s\n", buf,
                strerror(errno));
            exit(EX_UNAVAILABLE);
        }

        /* parent waits */
        if ((pid=waitpid(pid, &status, 0)) < 0) {
            fprintf(stderr, "shell: waitpid error: %s\n",
                strerror(errno));
        }
    }
}

```

\$EDITOR Examples: vim

Copy, yank, fold, markers, buffers etc.:

- set and display markers
(m[a-zA-Z], :marks)
- select visual blocks (v, V)
- format / indent selected block (=)
- delete, yank, use of buffers
(d, y, "xy, "xp)
- fold sections (zf, zA)
- autocomplete (^N)

```
Terminal — 80x38
/*
 * World's 2nd simplest shell
 * Same as simple-shell.c, but with a SIGINT signal handler.
 * Feed EOF (^D) to exit.
 *
 * Also illustrates forward declaration of a function prototype.
 */

#include <sys/types.h>
#include <sys/wait.h>

#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sysexits.h>
#include <unistd.h>

#include "getinput.h"
#include "sig.h"

int
main(int argc, char **argv) {
    char buf[BUFSIZ];
    pid_t pid;
    int status;

    /* cast to void to silence compiler warnings */
    (void)argc;
    (void)argv;

    if (signal(SIGINT, sig_int) == SIG_ERR) {
        fprintf(stderr, "signal error: %s\n", strerror(errno));
        exit(1);
    }
}

"simple-shell.c" 60L, 1212B
```


\$EDITOR Examples: vim

Integration with compiler, debugger, make(1) etc.:

- build your project (:make)
- show / go to error (:cc [N])
- jump to next / previous error (:cn, :cp)
- list errors (:cl, :copen)

```
Terminal — 80x38

/* cast to void to silence compiler warnings */
(void)argc;
(void)argv;

if (signal(SIGINT, sig_int) == SIG_ERR) {
    fprintf(stderr, "signal error: %s\n", strerror(errno));
    exit(1);
}

while (getinput(buf, sizeof(buf))) {
    buf[strlen(buf) - 1] = '\0';

    if (strncasecmp(buf, "exit", strlen(buf)) == 0) {
        exit(EXIT_SUCCESS);
    }

    if ((pid=fork()) == -1) {
        fprintf(stderr, "shell: can't fork: %s\n",
            strerror(errno));
        continue;
    } else if (pid == 0) { /* child */
        execlp(buf, buf, (char *)0);
        fprintf(stderr, "shell: couldn't exec %s: %s\n", buf,
            strerror(errno));
        exit(EX_UNAVAILABLE);
    }

    /* parent waits */
    if ((pid=waitpid(pid, &status, 0)) < 0) {
        fprintf(stderr, "shell: waitpid error: %s\n",
            strerror(errno));
    }
}

exit(EX_OK);
}
```


vi / vim graphical cheat sheet

Esc

normal mode

~ toggle case	! external filter	@ play macro	# prev ident	\$ eol	% goto match	^ "soft" bol	& repeat :s	* next ident	(begin sentence) end sentence	"soft" bol down	+ next line
\ goto mark	1	2	3	4	5	6	7	8	9	0 "hard" bol	- prev line	= auto ³ format
Q ex mode	W next word	E end word	R replace mode	T back 'till	Y yank line	U undo line	I insert at bol	O open above	P paste before	{ begin parag.	} end parag.	
q record macro	w next word	e end word	r replace char	t 'till	y yank ^{1,3}	u undo	i insert mode	o open below	p paste after ¹	[misc] misc	
A append at eol	S subst line	D delete to eol	F "back" find ch	G eof/ goto ln	H screen top	J join lines	K help	L screen bottom	: ex cmd line	" reg. ¹ spec	bol/ goto col	
a append	s subst char	d delete ^{1,3}	f find char	g extra ⁶ cmds	h ←	j ↓	k ↑	l →	: repeat t/T/f/F	' goto mk. bol	\ not used!	
Z quit ⁴	X back-space	C change to eol	V visual lines	B prev WORD	N prev (find)	M screen mid'l	< un- ³ indent	> indent ³	? find (rev.)			
Z extra ⁵ cmds	X delete char	c change ^{1,3}	V visual mode	b prev word	n next (find)	m set mark	reverse t/T/f/F	repeat cmd	/ find			

motion	moves the cursor, or defines the range for an operator
command	direct action command, if red , it enters insert mode
operator	requires a motion afterwards, operates between cursor & destination
extra	special functions, requires extra input

q commands with a dot need a char argument afterwards

bol = beginning of line, eol = end of line, mk = mark, yank = copy

words: `quux(foo, bar, baz);`
WORDS: `quux(foo, bar, baz);`

Main command line commands ('ex'):

:w (save), :q (quit), :q! (quit w/o saving)
:e f (open file f),
:%s/x/y/g (replace 'x' by 'y' filewide),
:h (help in vim), :new (new file in vim),

Other important commands:

CTRL-R: redo (vim),
CTRL-F/-B: page up/down,
CTRL-E/-Y: scroll line up/down,
CTRL-V: block-visual mode (vim only)

Visual mode:

Move around and type operator to act on selected region (vim only)

Notes:

- (1) use "x before a yank/paste/del command to use that register ('clipboard') (x=a..z,*) (e.g.: "ay\$ to copy rest of line to reg 'a')
- (2) type in a number before any action to repeat it that number of times (e.g.: 2p, d2w, 5i, d4j)
- (3) duplicate operator to act on current line (dd = delete line, >> = indent line)
- (4) ZZ to save & quit, ZQ to quit w/o saving
- (5) zt: scroll cursor to top, zb: bottom, zz: center
- (6) gg: top of file (vim only), gf: open file under cursor (vim only)

For a graphical vi/vim tutorial & more tips, go to www.viemu.com - home of ViEmu, vi/vim emulation for Microsoft Visual Studio

<https://duckduckgo.com/?q=vim+tutorial>

\$EDITOR

Other powerful Unix IDE integrations:

- code index and definitions via *e.g.*, `ctags(1)`
- a terminal multiplexer (*e.g.* `screen(1)` or `tmux(1)`)
- copious use of `Ctrl+Z` (*i.e.*, the shell's job control mechanisms)

See our tool tips for more details:

- <https://youtu.be/TWog5NklSws>
- <https://youtu.be/vxTXXaCr4s8>