Recent Developments

Bram Moolenaar
SFI Krakow April 2018
Vim popularity
Vim code size

code size over time (Kbyte) for src/*.[ch]
Impactful Changes

One has only limited time available to write code. What to work on next to be most impactful?
Impactful Changes

One has only limited time available to write code. What to work on next to be most impactful?

1. Work on a feature for 2-3 months tops
Impactful Changes

One has only limited time available to write code. What to work on next to be most impactful?

1. Work on a feature for 2-3 months tops

2. Build one feature on top of another one
Impactful Changes

One has only limited time available to write code. What to work on next to be most impactful?

1. Work on a feature for 2-3 months tops
2. Build one feature on top of another one
3. Understand the user
One has only limited time available to write code. What to work on next to be most impactful?

1. Work on a feature for 2-3 months tops
2. Build one feature on top of another one
3. Understand the user

So, how did this work for Vim?
Vim first releases

1991: Vim 1.14 distributed on Fish disk

Impactful changes in Vim 1.27:
- Port to Unix
- Port to MS-DOS

More supported systems == more users
Vim 2.0

Impactful changes:

- `:make` and error parsing

Please users: Efficient edit - build - fix cycles
Vim 2.0

Impactful changes:

- `:make` and error parsing

Please users: Efficient edit - build - fix cycles

- Vi compatibility

Please distributors (Vim on every Linux and Mac system)
Vim 3.0

Impactful changes:

- Multiple windows and buffers
  Make use of larger screens and more memory
Vim 3.0

Impactful changes:

- Multiple windows and buffers
  Make use of larger screens and more memory

- Swap file
  Reliability, user trust
Vim 4.0

Impactful changes:

- Help in a window

More complexity requires more support
Vim 4.0

Impactful changes:

- Help in a window
  
  More complexity requires more support

- Autocommands
  
  Extensibility, you can’t build everything yourself
Vim 4.0

Impactful changes:

- Help in a window
  More complexity requires more support

- Autocommands
  Extensibility, you can’t build everything yourself

- MS-Windows port
  Reality: most users are there
Vim 5.0

Impactful changes:

- Syntax highlighting

Making use of faster computers and better screens
Impactful changes:

- Syntax highlighting
  Making use of faster computers and better screens

- Vim script
  Extensibility, you can’t build everything yourself
Vim 6.0

Impactful changes:

- Unicode support

ASCII is no longer the standard
Vim 6.0

Impactful changes:

● Unicode support
  ASCII is no longer the standard

● Automatic indenting
  Users are lazy
Impactful changes:

- Unicode support
  ASCII is no longer the standard

- Automatic indenting
  Users are lazy

- Plugins
  Extensibility, you can’t build everything yourself
Vim 7.x

Impactful changes:

- Persistent undo

Don’t worry, you can go back in time (and back to the future)
Impactful changes:

- Jobs, Channels and Timers

Extensibility, you can’t build everything yourself
Make use of more powerful computers
Vim 8.1

Impactful changes?
Vim 8.1

Impactful changes:

- Terminal window

Why?
Impactful changes:

- Terminal window

Why?

I want to debug Vim over ssh, which requires:
1. Window running gdb
2. Window running program being debugged
3. Window(s) to edit source code
Terminal debugger

```
new-ui mi /dev/pts/15
Reading symbols from vim...done.
(gdb) new-ui mi /dev/pts/15
New UI allocated
(gdb) break ex_help
Breakpoint 1 at 0x99840: file ex_cmds.c, line 6194.
(gdb) run
Starting program: /home/mool/vim/vim80/src/vim
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Breakpoint 1, ex_help (eap=0x7fffffffdf4f0) at ex_cmds.c:6194
   6194 { (gdb)
```

```
void
ex_help(exarg_T *eap)
{
    char_u *arg;
    char_u *tag;
    FILE *helpfd; /* file descriptor of help file */
    int n;
    int i;
    win_T *wp;
    int num_matches;
    char_u **m;
    char_u *p;
    eap: 0x7fffffffdf4f0 = {
        int emp arg = 0x555555dcb303 "gui",
        int alt nextcmd = 0x0,
        buf_T *bu cmd = 0x555555dcb300 "he gui",
        #ifdef FEAT_MULTI_L
        cmdlinep = 0x7fffffffdf710,
        #endif
        char_u *la argt = 2954,
        skip = 0,
        #ifdef FEAT_FOLDING
        forceit = 0,
        old addr_count = 0,
        line1 = 1,
        line2 = 1,
        #endif
        if (eap != NULL)
        {
            /* A "help
            * followed append = 0,
            * userfilter = 0,
            */
            for (arg = eap->arg; *arg; ++arg)
            {
                if (*arg == '\n' || *arg == '\r'
                {
                    *arg++ = NUL;
                    eap->nextcmd = arg;
                }
```

```
```
Terminal debugger

Parts needed:
1. Terminal emulator: libvterm
2. Jobs and channels
3. Window toolbar for Step/Next/Continue...
4. Balloon to show variable values
5. Popup menu
Terminal window

How the parts are put together:

Vim

- Window Toolbar
- Balloon
- Popup menu

libvterm

Job

Channel
Terminal window

Vim

1. read stdin
2. Decode K_DOWN
3. Write
4. VTERM_KEY_DOWN

Vterm

2. Convert key to
   <Esc>OB
3. Read bytes
4. Send bytes to Job
5. (wait)
6. Parse bytes
7. Update virtual screen
8. Invoke callbacks
9. (work)
10. (wait)
11. Parse bytes
12. Invoke cursor position callback

Job

4. Read cursor-down
5. (work)
6. Write screen updates
7. (work)
8. Write cursor move
9. <Esc>[20;30H
Terminal debugger

Debugger demo
Terminal debugger

new-ui mi /dev/pts/16
Reading symbols from vim...done.
(gdb) new-ui mi /dev/pts/16
New UI allocated
(gdb) 

!gdb [running] 0,0-1 All

gdb program [active] 0,0-1 All [No Name] 0,0-1 All
Terminal debugger

new-ui mi /dev/pts/17
Reading symbols from vim...done.
(gdb) new-ui mi /dev/pts/17
New UI allocated
(gdb) run
Starting program: /home/mool/vim/vim80/src/vim
[Thread debugging using libthread_db enabled]
Using host libthread_db library "libc-x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, ex help (eap=0x7ffef24d4f0) at ex_cmds.c:6215
6215    if (eap != NULL)
(gdb)

VIM - Vi IMproved
version 8.0.1615
by Bram Moolenaar et al.
Vim is open source and freely distributable

Become a registered VIm user!
type :help register<Enter> for information

type :q<Enter> to exit
 type :help<Enter> or <F1> for on-line help
 type :help version8<Enter> for version info

if (eap != NULL)
Terminal debugger
Terminal window

What else can you use it for?
Terminal window

Run make in a terminal window
Terminal window

Running external command in the GUI

```c
#define ANSI

color ramp

Press ENTER or type command to continue
```
Terminal window

Testing with a screenshot diff
Testing old style

.in file:

STARTTEST
:so small.vim
:set belloff=all
/Start cursor here
vaBiBD:?Bug?,/Piece/-2w! test.out
/^- Bug
:s/u/~u~/
:s/i/~u~/
:s/o/~~~/
:.w >>test.out

.ok file:

- Bug in "vPPPP" on this text (Webb):
  {
  }
- Bug uuun "vPPPP" uuuuuuuuuun this text (Webb):
Testing new style

.vim file:

```vim
func Test_move_cursor()
    new
call setline(1, ['aaa', 'bbb', 'ccc', 'ddd'])
call cursor([1, 1, 0, 1])
call assert_equal([1, 1, 0, 1], getcurpos()[1:])
call cursor([4, 3, 0, 3])
call assert_equal([4, 3, 0, 3], getcurpos()[1:])
call cursor(2, 2)
call assert_equal([2, 2, 0, 2], getcurpos()[1:])
" line number zero keeps the line number
call cursor(0, 1)
call assert_equal([2, 1, 0, 1], getcurpos()[1:])
```
Testing with a screenshot

.vim file:

```vim
func Test_popup_position()
   if !CanRunVimInTerminal()
      return
   endif
   call writefile(['123456789_123456789_123456789_a', '123456789_123456789_123456789_b', '123', '], 'Xtest')
   let buf = RunVimInTerminal('Xtest', {})
   call term_sendkeys(buf, ":vsplit<CR>"

   " default pumwidth in left window: overlap in right window
   call term_sendkeys(buf, "GA<C-N>")
   call VerifyScreenDump(buf, 'Test_popup_position_01', {'rows': 8})
```
Terminal window

Testing with a screenshot diff
Vim 8.1

Release: “in a few weeks”
The end

Questions?