



# Pengantar Vi iMproved

Sebuah panduan praktikal Vim sebagai  
editor teks sehari-hari

Muhamad Nauval Azhar

# Hak Cipta

## Pengantar Vi iMproved

Hak Cipta © 2022 oleh Muhamad Nauval Azhar

Semua hak cipta untuk buku ini dimiliki oleh Muhamad Nauval Azhar. Buku ini diterbitkan secara independen oleh Muhamad Nauval Azhar. Segala bentuk penggandaan, penyebaran, atau penggunaan untuk tujuan komersial tanpa izin tertulis dari pengarang adalah dilarang.

Anda dapat mengunduh salinan buku ini untuk keperluan pribadi atau pendidikan, namun Anda tidak diizinkan untuk menjual atau mendistribusikan ulang salinan buku ini tanpa izin tertulis dari pengarang. Selain itu, Anda juga tidak diizinkan untuk memodifikasi, memperbanyak, atau menggunakan konten buku ini untuk tujuan komersial tanpa izin tertulis dari pengarang.

# Daftar Isi

<i>Hak Cipta .....</i>	<b>1</b>
<i>Daftar Isi.....</i>	<b>2</b>
<i>Kata Pengantar.....</i>	<b>6</b>
<i>Konvensi .....</i>	<b>10</b>
<i>Prolog .....</i>	<b>12</b>
<i>Alasan Menggunakan Vim.....</i>	<b>13</b>
VS Code Lambat.....	13
Lebih Produktif.....	14
<i>Tentang Vim.....</i>	<b>23</b>
<i>Sejarah Singkat.....</i>	<b>24</b>
<i>Neovim .....</i>	<b>26</b>
<i>Konfigurasi .....</i>	<b>27</b>
<i>Case Sensitive.....</i>	<b>29</b>
<i>Memasang Vim.....</i>	<b>30</b>
<i>macOS .....</i>	<b>30</b>
<i>Windows .....</i>	<b>34</b>
Windows Subsystem Linux .....	41
<i>Unix-like.....</i>	<b>48</b>
<i>Mempelajari Vim .....</i>	<b>51</b>
<i>Membuka Dan Keluar Dari Vim .....</i>	<b>51</b>
<i>Membuka Berkas Dengan Vim .....</i>	<b>54</b>
<i>Mengetik Kode di Vim .....</i>	<b>56</b>
<i>Mengatur Syntax Highlighter.....</i>	<b>59</b>
<i>Menyimpan Berkas .....</i>	<b>61</b>
<i>Menampilkan Nomor Baris .....</i>	<b>63</b>
<i>Mengatur Colorscheme.....</i>	<b>65</b>

<b>Motion.....</b>	<b>66</b>
Word Motion.....	68
Text Object Motion.....	69
Text Object Selection.....	72
Jump Motion.....	73
<b>Operator .....</b>	<b>74</b>
<b>Copy dan Paste.....</b>	<b>78</b>
<b>Undo, Redo dan Repeat.....</b>	<b>78</b>
<b>Search, Substitute, dan Replace .....</b>	<b>80</b>
<b>Visual Mode .....</b>	<b>85</b>
<b>Lompat-Melompat.....</b>	<b>91</b>
<b>Indentasi.....</b>	<b>92</b>
<b>Konfigurasi .....</b>	<b>98</b>
<b>Plugin.....</b>	<b>102</b>
<b>Buffer .....</b>	<b>108</b>
<b>Windows .....</b>	<b>115</b>
<b>Tab .....</b>	<b>120</b>
<b>Registers .....</b>	<b>123</b>
<b>Macros .....</b>	<b>126</b>
<b>External Command.....</b>	<b>129</b>
<b>Command-line Mode.....</b>	<b>132</b>
<b>Neovim.....</b>	<b>135</b>
<b>Memasang Neovim.....</b>	<b>137</b>
macOS .....	137
<b>Windows .....</b>	<b>138</b>
Memasang Scoop.....	138
Memasang Neovim.....	139
Memasang Git.....	140
Memasang VC Redist.....	144
<b>Unix-like.....</b>	<b>147</b>

Snapcraft.....	147
APT.....	148
<b>Konfigurasi Neovim.....</b>	<b>149</b>
<b>AstroNvim .....</b>	<b>152</b>
<b>Memasang AstroNvim.....</b>	<b>154</b>
macOS .....	156
Memasang Nerd Fonts.....	158
Windows .....	161
Memasang Nerd Fonts.....	166
Unix-like.....	171
Memasang Nerd Fonts.....	179
<b>Penggunaan .....</b>	<b>183</b>
Antarmuka .....	183
<b>Panduan Dasar .....</b>	<b>185</b>
Umum .....	189
Bufferline.....	190
Neo-Tree.....	190
Dashboard.....	190
Session Manager .....	190
Package Management .....	191
Language Server Protocol.....	191
Telescope.....	192
Toggle Terminal .....	193
Git .....	193
User Interface.....	193
<b>Konfigurasi .....</b>	<b>194</b>
Kustomisasi.....	196
Dashboard.....	198
Colorscheme .....	201
Statusline.....	205
Plugin .....	208
<b>Syntax Highlighter &amp; Language Server Protocol.....</b>	<b>212</b>
Syntax Highlighter.....	212
Language Server Protocol.....	215
<b>Penutup.....</b>	<b>222</b>

*Penulis.....*..... 224

# Kata Pengantar

Sebagai pengarjin kode, menguasai alat editor teks merupakan suatu kewajiban. Umumnya, mereka memiliki preferensi sendiri dalam mengatur editor teks sedemikian rupa sehingga dapat digunakan dengan nyaman dan meningkatkan produktivitas dalam mengembangkan perangkat lunak.

Tedapat banyak sekali editor teks modern di luar sana, editor teks tersebut datang dan pergi, seolah memiliki masanya masing-masing. Sekitar satu dekade yang lalu saya masih nyaman menggunakan editor teks seperti Sublime Text, namun sekarang seperti sudah tidak relevan lagi.

Sekarang masanya editor teks seperti VS Code, editor teks ini *open source* dan memiliki komunitas yang besar, tidak heran banyak sekali yang menggunakannya. Selain VS Code, banyak lagi editor teks lain berbayar yang memiliki fitur-fitur *premium*.

Memilih editor teks adalah persoalan subjektif: mungkin kita memilihnya karena itu populer, atau mungkin kita memilihnya karena memiliki fitur yang ciamik. Semua itu persoalan preferensi yang tidak pernah bisa diperdebatkan, namun dihargai.

Hal yang membuat kita bertahan pada sebuah editor teks biasanya soal kenyamanan dan *muscle memory*. Saat di dalam editor teks tersebut kita melakukan penyuntingan kode dengan mudah tanpa memikirkan setiap langkah untuk melakukannya.

Ketika kita memutuskan beralih kepada editor teks yang lain karena beberapa alasan, hal yang menyulitkan adalah persoalan pemetaan tombol. Kita sudah terbiasa dengan pemetaan tombol pada editor teks sebelumnya. Namun, sial, editor teks yang baru ini memiliki pemetaan yang berbeda.

Saya pernah berada pada situasi yang sama, ketika saya merasa Sublime Text sudah kadaluarsa, saya memutuskan untuk beralih pada VS

Code yang saat itu memiliki fitur lebih mutakhir. Tentu saja saya mengalami kesulitan pada kesan pertama, namun itu konsekuensi logis yang perlu saya ambil untuk mencapai kemaslahatan.

Seiring berjalannya waktu, saya pun terbiasa dengan pemetaan-pemetaan yang baru hingga menjadi *muscle memory*, dan pada akhirnya saya merasakan pengalaman yang lebih baik dalam melakukan pengembangan perangkat lunak.

Pada VS Code, semakin banyak ekstensi yang dipasang semakin banyak juga sumber daya yang dikonsumsi. Komputer saya tidak pernah memiliki memori yang lebih dari 4 GB saat itu. Sehingga saya perlu mencari alternatif lain.

Urgensitas itulah yang menghadirkan ide radikal untuk beralih ke Vim. Tentu saja beralih dari editor teks konvensional ke *modal* editor seperti Vim lebih sulit lagi. Banyak hal yang perlu dipelajari, terutama mebiasakan diri dengan beberapa mode di dalamnya.

Bagi saya, mempelajari Vim saat itu adalah salah satu keputusan terbaik sepanjang masa walaupun membutuhkan waktu yang relatif lama untuk memahaminya secara mendasar.

Saat pertama kali membuka Vim, saya tidak tahu bagaimana cara untuk keluar dari *program* tersebut. Sekarang, saat saya sudah cukup memahaminya, saya menyengaja untuk tidak pernah keluar.

Vim juga merupakan sebuah editor teks yang tidak pernah mati, seolah ia berada di semesta yang berbeda. Ketika banyak orang tergiur dengan editor teks modern yang penuh dengan magis, para pengguna Vim tidak berpaling.

Pengantar Vi iMproved adalah buku yang ditujukan bagi mereka yang ingin mempelajari Vim sebagai pilihan alternatif editor teks. Dengan mempelajari Vim, kita akan dapat menyunting teks dengan lebih cepat dan efisien, dan memanfaatkan kekuatan editor tersebut untuk meningkatkan produktivitas.

Buku ini dimaksudkan untuk membantu pembaca memahami dasar-dasar Vim, mulai dari mengenal editor ini hingga menguasai perintah dan fiturnya.

Selain itu, buku ini terdiri dari banyak bagian, namun secara garis besar buku ini akan membahas Vim, Neovim dan konfigurasinya dengan AstroNvim untuk dapat dijadikan editor teks sehari-hari.

Setiap bagian menggunakan pendekatan yang berbeda. Pada bagian Vim, kita akan mempelajari Vim dengan pendekatan filosofis, bukan dengan membaca daftar perintah dan menghafalnya. Melainkan memahaminya secara mendasar, memahami perintah-perintah sebagai bahasa komunikasi dengan Vim.

Menghafal perintah-perintah mudah saja dilakukan, tapi dengan memahaminya secara mendasar, kita dapat mengerti bagaimana perintah tersebut dapat terstruktur menjadi satu perintah yang utuh dan bagaimana perintah-perintah tersebut dapat digabungkan untuk mencapai suatu aksi tertentu.

Buku ini tidak mencakup semua hal di dalam Vim, karena tujuannya bukan untuk menulis ulang dokumentasi Vim dalam Bahasa Indonesia. Tapi, buku ini mencakup konsep-konsep penting dalam Vim, seperti mode editor, *operator* dan *motion*, serta cara membuat konfigurasi Vim yang sesuai dengan kebutuhan.

Setelah mempelajari Vim, kita akan beralih menggunakan Neovim sebagai editor teks sehari-hari. Bagian ini dan bagian terusannya akan menggunakan pendekatan yang pragmatis. Kita tidak akan memahaminya secara mendalam hingga menjadi ahli Neovim, melainkan hanya akan menjadikannya alat untuk membantu kita dalam pengembangan perangkat lunak.

Pada dasarnya, antara Vim dan Neovim memiliki kesamaan, karena Neovim sendiri adalah proyek *fork* dari Vim. Saat kita sudah memahami Vim, kita tidak perlu mempelajari Neovim dari awal lagi.

Buku ini dapat dijadikan referensi bagi mereka yang ingin mencoba alternatif lain dalam menyunting kode, memahami Vim secara men-dasar, dan mulai berkelana di ekosistem Vim. Ada alasan mengapa buku ini diberi judul "Pengantar Vi iMproved".

Kita perlu membiasakan mencoba alternatif lain untuk melihat potensi pengalaman yang lebih baik. Bukan menutup diri dengan ber-lagak konservatif.

Buku ini saya dedikasikan untuk para pengembang yang hendak mempelajari Vim, terutama para penikmat video tutorial saya di YouTube yang seringkali protes ketika saya menggunakan Vim.

Selamat mempelajari Vim!

# Konvensi

Teks yang ditulis dengan *font monospace* akan merepresentasikan kode keseluruhan, alamat berkas/folder, bagian kode, dan perintah-perintah Vim atau CLI. Selain kode, penulisan teks dengan *font monospace* juga diterapkan untuk merepresentasikan teks yang berada di dalam Vim (tidak semua contoh menggunakan bahasa pemrograman, kadang-kadang menggunakan teks biasa), contoh penulisannya seperti ini:

```
  Lorem ipsum dolor sit amet, consectetur adipiscing
  elit. Proin semper convallis dolor, in bibendum nulla
  laoreet ac. Fusce rutrum nisi ac ultrices iaculis. Sed
  imperdiet nisi quis ligula semper euismod.
```

Contoh penulisan blok kode *program*:

```
function add(a, b) {
  return a + b;
}

console.log(add(1, 2));
```

Tidak semua yang ditulis pada blok kode seperti di atas adalah kode *program*, kadang-kadang hasil dari perintah CLI atau Vim ditulis dengan konvensi yang sama, seperti ini:

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

Posisi kursor pada Vim direpresentasikan dengan karakter yang diberi warna latar belakang merah dan warna teks putih (bila kamu mendapatkan buku ini dalam versi cetak hitam-putih, warna latar belakang akan menjadi hitam):

```
console.log("Hello");
```

Contoh di atas menunjukkan bahwa posisi kursor berada di huruf s pada bagian kode `console`. Bila posisi kursor berada karakter kosong, seperti spasi atau baris baru maka akan direpresentasikan dengan karakter |, seperti ini:

```
// kode bagian atas  
|  
// kode bagian bawah
```

Penulisan URL, istilah Vim, nama berkas atau ekstensi dari sebuah berkas akan ditulis dalam huruf miring, seperti *welcome.js*, *bufferline*, *vimrc*, atau <https://vim.org>.

Setiap perintah yang ditulis di buku ini pada dasarnya untuk pengguna Unix atau Unix-*like*, seperti macOS, Linux, atau WSL. Perintah yang berbeda untuk sistem operasi Windows akan dituliskan setelahnya dengan kalimat semacam "Untuk Windows ...".

Penyebutan Windows di buku ini merujuk pada *program gVim* yang akan dibahas pada bagian berikutnya, pengguna Windows dengan WSL diklasifikasikan menjadi pengguna Linux. Namun, untuk beberapa kasus tertentu tetap perlu mengikuti panduan untuk Windows, seperti pengaturan *font* salah satunya, karena *terminal emulator* yang digunakan untuk menjalankan WSL berjalan pada lingkungan Windows.

# Prolog

Sekitar 2 tahun lalu saya mulai menjadikan Vim sebagai editor teks sehari-hari. Sebelum itu, saya udah mencoba pelbagai editor teks, mulai dari Notepad, Sublime Text, Atom, VS Code, bahkan Dreamweaver pun saya sudah pernah coba.

Pada dasarnya mereka semua sama saja, sama-sama editor teks. Yang membedakan satu dengan yang lain hanyalah fitur, *keybinding*, dan komunitas. Dari beberapa editor teks yang saya sebutkan tadi, VS Code yang paling banyak digunakan sekarang<sup>1</sup>. Sisanya berusaha untuk menjadi relevan dan bahkan Atom sudah mau “pensiun” akhir tahun ini<sup>2</sup>.

Yang membuat VS Code ini banyak peminat adalah selain gratis ia juga memiliki komunitas yang besar, itu yang membuat editor teks ini banyak memiliki pilihan ekstensi dan membuat para *coder* jadi jatuh cinta.

Selain yang saya sebutkan tadi, tentunya masih banyak editor teks di luar sana, ya! Apalagi produknya JetBrains~

*Menggunakan editor teks jenis apa saja tidak masalah, selama editor teks tersebut dapat membuatmu lebih produktif*

---

1 "2022 Developer Survey" <https://survey.stackoverflow.co/2022/#most-loved-dreaded-and-wanted-new-collab-tools-love-dread>

2 "Sunsetting Atom" <https://github.blog/2022-06-08-sunsetting-atom/>

# Alasan Menggunakan Vim

Setelah berkelana mencoba beberapa editor teks tadi, akhirnya saya memutuskan untuk belajar dan menggunakan Vim sampai saat ini. Alasannya sederhana saja: ringan dan lebih produktif.

## VS Code Lambat

Dua editor teks terakhir yang saya gunakan sebelum Vim adalah Sublime Text dan VS Code. Sublime Text adalah editor teks yang saya gunakan sejak saya masih sekolah. Sampai akhirnya saya beralih ke VS Code karena banyak *package-package* dari Sublime Text yang mulai kadaluarsa dan tidak dikembangkan lagi.

VS Code memiliki komunitas yang besar dan aktif, sehingga banyak sekali ekstensi-ekstensi yang bisa kita gunakan. Namun, yang tidak saya sadari saat itu adalah semakin banyak ekstensi yang digunakan maka semakin banyak sumber daya komputer yang dibutuhkan.

Pada titik itulah VS Code mulai terasa lebih lambat. Tanpa menggunakan ekstensi yang banyak pun, sebetulnya VS Code sudah cukup lambat pada komputer saya saat itu.

Alasan yang paling masuk akal kenapa VS Code lambat adalah karena berbasis web – di dalamnya terdapat *browser*, sehingga butuh sumber daya yang cukup banyak dan komputer saya tidak bisa memberikannya.

Itu alasan pertama dan yang terpenting kenapa saya akhirnya memutuskan untuk beralih ke Vim. Tentu saja Vim lebih cepat ketika dibuka dan lebih responsif. Saya hanya membutuhkan *terminal emulator* dan langsung bisa membuka program editor teks bernama Vim tersebut.

Jujur saja, saya tidak memiliki alasan lain yang lebih *urgent* dari alasan sebelumnya. Dipikiran saya saat itu hanyalah bagaimana caranya saya masih bisa *ngoding* dengan performa yang lebih *ngebut*. Saat itulah

saya bertemu dengan Vim, menyesatkan diri ke Vim, dan ingin terus bersama Vim~

*Bila menggunakan Vim adalah salah, maka selamanya saya tidak ingin benar*

Saya tidak punya *benchmark* untuk seberapa cepat Vim, tapi setidaknya ia lebih cepat dari VS Code di komputer saya. Ini sudah pasti kedengarannya subjektif, tapi memilih editor teks pada dasarnya tidak lebih dari melibatkan perasaan saja. Mana yang dirasa cocok itu yang akan dipilih.

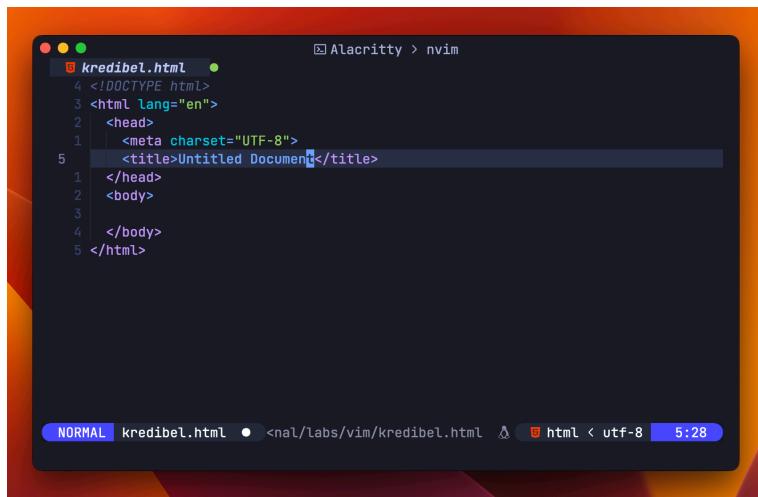
## **Lebih Produktif**

Ketika sudah tersesat di Vim, akhirnya saya mulai terbiasa dengan Vim dan bagaimana cara mengoperasikannya. Dari sinilah saya baru merasakan bahwa ternyata ngoding bisa lebih cepat dengan Vim.

Bagaimana tidak, pada saat saya menggunakan editor teks semacam VS Code, Sublime Text dan semacamnya, saya masih sering menggunakan *mouse*. Tangan saya bolak-balik dari atas *keyboard* ke atas *mouse* dan itu berulang. Sedangkan dengan Vim semuanya bisa selesai dengan perintah-perintah yang saya ketik di atas *keyboard*.

Hal ini yang saya baru sadari ternyata menggunakan *mouse* pada saat *ngoding* itu cukup membuang waktu walaupun hanya sekitar beberapa detik saja. Tapi tetap saja artinya ada sekian detik yang terbuang. Lagipula untuk apa pindah ke *mouse* apabila bisa diselesaikan di atas *keyboard*?

Sebagai contoh, saya ingin mengganti teks konten dari *tag* HTML `<title>` dari “Untitled Document” menjadi “Kredibel” dan posisi kursor saat itu ada di atas tanda “`<`” pada pembuka *tag* `<title>`.



```
Alacritty > nvim
kredibel.html
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5       <title>Untitled Document</title>
6   </head>
7   <body>
8
9   </body>
10 </html>

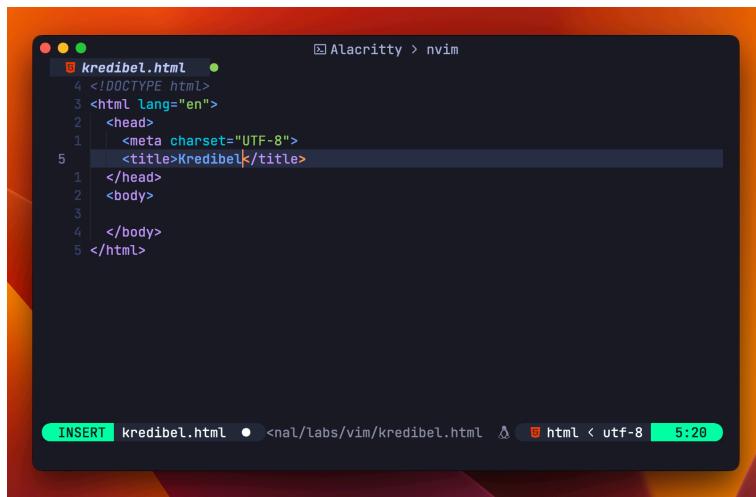
NORMAL kredibel.html • <nal/labs/vim/kredibel.html ▲ & html < utf-8 5:28
```

Contoh kode *study* kasus Vim

Umumnya, di editor teks seperti VS Code, saya setidaknya perlu melakukan:

- Menggeser kursor ke posisi di antara tag `<title>`, entah menggunakan arah panah *keyboard* atau menggeser *mouse*
- Memblok dan menghapus isi teks “Untitled Document”
- Menulis teks “Kredibel”

Tapi, di Vim saya hanya perlu menulis perintah `cit` dan menulis “Kredibel”, selesai.

A screenshot of the nvim text editor running in Alacritty terminal. The window title is 'kredibel.html'. The code in the buffer is:

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="UTF-8">
5       <title>Kredibel</title>
6   </head>
7   <body>
8
9 </body>
10 </html>
```

The cursor is positioned at the end of the word 'Kredibel' in the title tag. The status bar at the bottom shows 'INSERT' mode, the file name 'kredibel.html', the path '/nial/labs/vim/kredibel.html', the encoding 'html', the character set 'utf-8', and the time '5:20'.

Vim *changes inner tag*

Contoh di atas hanyalah satu hal dari sekian banyak hal yang bisa kita lakukan lebih cepat dengan Vim.

Perintah `cit` di Vim artinya *change-inner-tag-block*. Dan Vim juga paham konteks (*tag*) yang dimaksud. Karena posisi kursor saya pada saat memberikan perintah `cit` berada di atas tanda "<" tag `<title>`. Maka tag yang dimaksud adalah `<title>` tersebut.

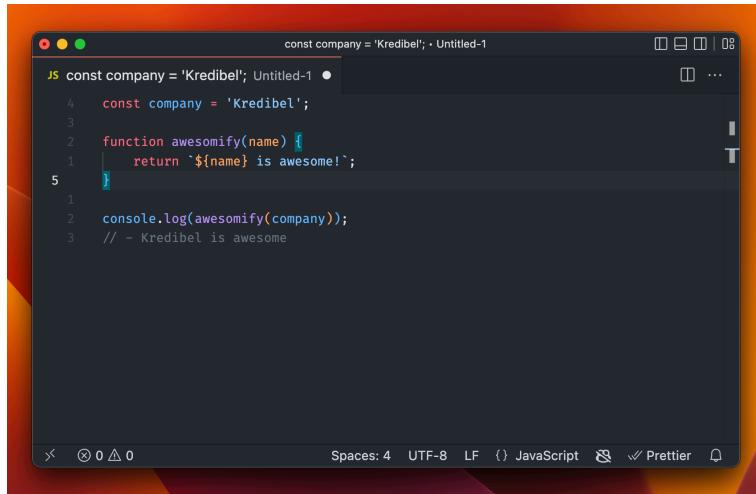
Mungkin kamu bertanya-tanya, bukankah bila mengetik `cit` di *keyboard* akan memunculkan tulisan "cit" secara harfiah di layar? Hal itu benar bila di editor teks *non-modal* seperti VS Code. Tapi, di Vim tidak seperti itu, karena Vim ini termasuk ke *modal editor*.

*Modal editor* merupakan sebuah editor yang memiliki beberapa mode. Sebagai contoh Vim, editor teks ini memiliki beberapa mode di antaranya: *normal*, *insert*, *visual*, *command*, dan beberapa mode tambahan lainnya.

Berbeda dengan editor teks konvensional seperti VS Code, Sublime Text, Atom, Emacs, atau Nano mereka termasuk ke *non-modal editor*, karena hanya memiliki satu mode saja yaitu *insert mode*.

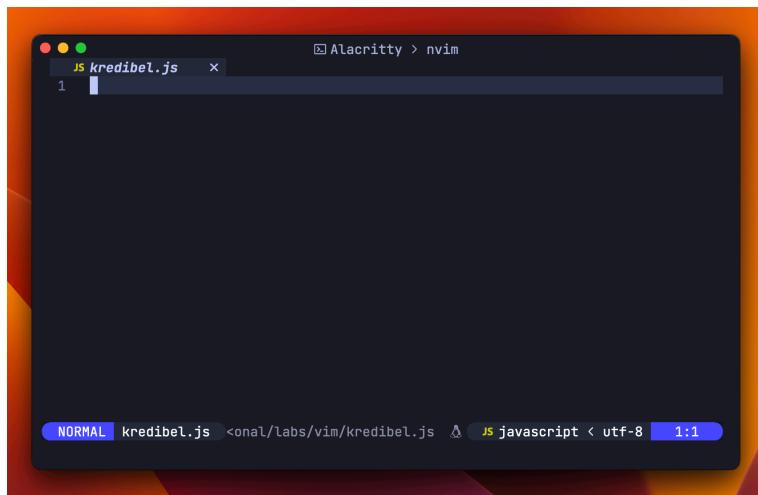
Sudah kelihatan perbedaan signifikan dari kedua jenis editor ini, kan? Ya, masalah mode!

Begini, umumnya bila kita menggunakan editor teks seperti VS Code atau sejenisnya, ketika kita buka editor teks tersebut, kita sudah bisa langsung mengetik apa saja yang kita mau.



Pratinjau Editor Teks Visual Studio Code

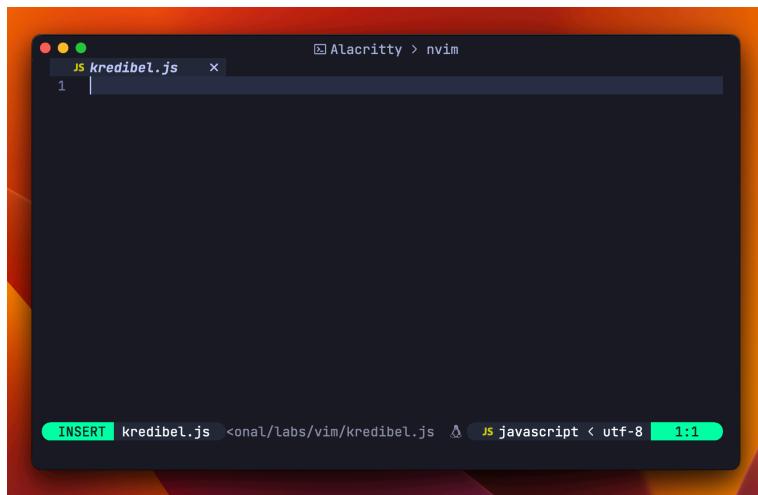
Sedangkan pada *modal editor* seperti Vi, Vim, atau Neovim, ketika kita membuka editor tersebut, umumnya kita tidak bisa langsung mengetik teks apapun yang kita inginkan. Ini disebabkan oleh editor tersebut diawali oleh mode normal.



Pratinjau Vim

Kamu lihat tulisan “NORMAL” di pojok-kiri-bawah? Ya, itu adalah indikator mode dari Vim. Pada mode normal ini, kita tidak bisa mengetik tulisan apapun di dalamnya, bila kita mencoba menekan tombol di *keyboard* maka Vim akan menerjemahkannya sebagai perintah atau *command* ketimbang memunculkan karakter yang kita tekan.

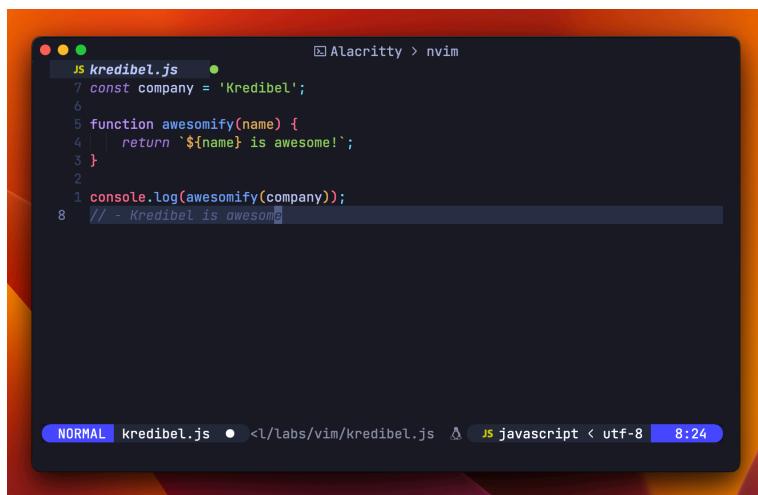
Sebagai contoh, bila kita tekan tombol `i` di *keyboard*, ini akan memunculkan huruf “`i`” secara harfiah pada editor teks *non-modal* seperti VS Code. Tapi, bila kita lakukan hal yang sama di dalam Vim, maka tidak akan muncul huruf “`i`” di layar, yang terjadi adalah Vim akan berganti mode dari normal menjadi *insert mode*.



A screenshot of a terminal window titled "Alacritty > nvim". The window shows a single line of code: "1 |". The status bar at the bottom indicates "INSERT kredibel.js <onal/labs/vim/kredibel.js & js javascript < utf-8 1:1".

Vim mode insert

Karena *i* diterjemahkan sebagai perintah berganti mode menjadi *insert* oleh Vim. Di mode inilah kita bisa mengetik apapun seperti yang kita bisa lakukan di editor teks *non-modal*.



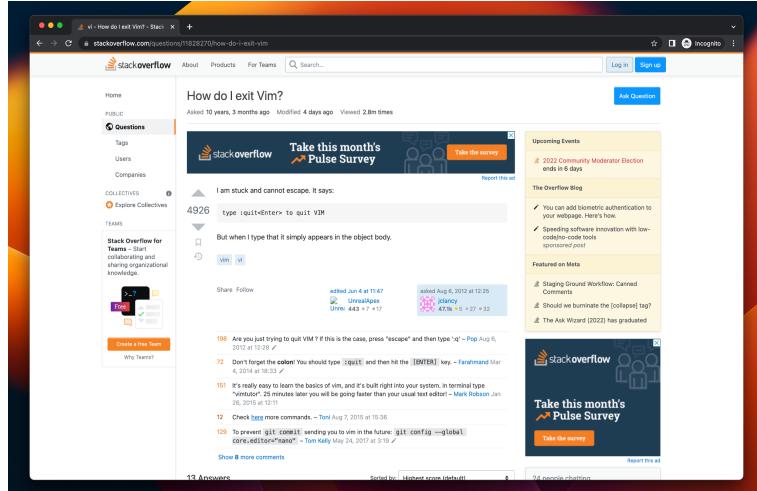
A screenshot of a terminal window titled "Alacritty > nvim". The window shows the following JavaScript code:

```
1 const company = 'Kredibel';
2
3 function awesomify(name) {
4   return `${name} is awesome!`;
5 }
6
7 console.log(awesomify(company));
8 // - Kredibel is awesome
```

The status bar at the bottom indicates "NORMAL kredibel.js ● <onal/labs/vim/kredibel.js & js javascript < utf-8 8:24".

Vim mode insert dengan kode JavaScript di dalamnya

Bila ingin kembali ke mode normal, kita bisa menekan tombol **ESCAPE** di *keyboard*. Dan pertanyaan yang paling sering ditanyakan pada Vim: bagaimana cara saya keluar dari Vim?<sup>3</sup>



Pertanyaan di situs Stack Overflow mengenai cara keluar dari program Vim

Untuk keluar dari Vim itu sederhana saja: *restart computer* dan Vim akan keluar dengan sendirinya. Becanda, ya!

Tapi, serius, untuk keluar dari Vim ini memang sederhana, cukup tulis perintah :q di *command mode* maka kita akan keluar dari Vim. Untuk melakukannya:

- Pertama, tekan tombol **SHIFT+;** untuk mencapai tanda titik dua
- Sampai sini sudah ada di dalam *command mode*
- Lalu tekan tombol **q** dan tekan **ENTER**

Bila kita memiliki tulisan di dalam Vim yang sebelumnya kita ketik, maka Vim tidak akan keluar begitu saja dengan perintah sebelumnya.

---

<sup>3</sup> "How do I exit Vim?" <https://stackoverflow.com/questions/11828270/how-do-i-exit-vim>

A screenshot of a terminal window titled "Alacritty > nvim". The terminal shows a file named "kredibel.js" with the following content:

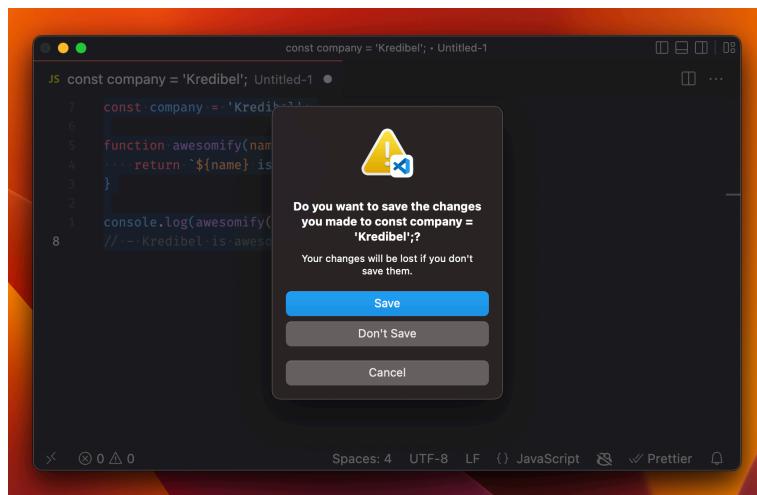
```
JS kredibel.js
7 const company = 'Kredibel';
6
5 function awesomify(name) {
4   return `${name} is awesome!`;
3 }
2
1 console.log(awesomify(company));
8 // - Kredibel is awesome
```

At the bottom of the terminal, there is a message box containing the following error messages:

E37: No write since last change  
E162: No write since last change for buffer "kredibel.js"  
Press ENTER or type command to continue

*Error Vim karena perubahan belum disimpan*

*Error tersebut setara dengan dialog peringatan pada non-modal editor.*



*Dialog peringatan berkas belum disimpan pada VS Code*

Perbedaannya adalah di Vim tidak ada tombol sama sekali, hanya ada tulisan *error*. Ketika muncul *error* seperti itu di Vim, setidaknya kita memiliki dua pilihan: simpan perubahan dan keluar; atau membatalkan perubahan dan langsung keluar.

Untuk memilih opsi yang pertama, kita bisa menggunakan perintah `:wq` di Vim yang bermaksud *write-(and)-quit*. Dan bila kita ingin opsi yang kedua, cukup tambahkan tanda seru diakhir perintah *quit*, jadi `:q!.`

Keduanya akan membuat kita keluar dari Vim, yang membedakan hanyalah kita ingin menyimpan atau tidak perubahan yang sudah dibuat.

Sampai sejauh ini mungkin kamu sudah merasakan betapa “ribet”-nya Vim ini!

Saya ingin jujur, sebenarnya kita memiliki reaksi serupa pada saat pertama kali berinteraksi dengan Vim. Bagaimana tidak, di editor teks seperti VS Code itu lebih mudah karena kita tinggal tekan-tekan tombol maka semuanya selesai. Di Vim, kita harus ketik-ini-ketik-itu baru selesai. Karena semua interaksinya dilakukan dengan *keyboard*.

Tapi, itulah poin penting menggunakan Vim, membuat tangan kita tidak berpindah dari atas *keyboard* ke *mouse*. Percayalah itu hanya soal kebiasaan saja, sekali kita sudah terbiasa dengan Vim maka semuanya akan lebih *ngebut!*

# Tentang Vim

Vim adalah program editor teks yang dapat dijalankan dalam CLI maupun GUI. Umumnya orang menggunakan Vim di dalam lingkup CLI melalui *terminal emulator*<sup>4</sup>, seperti iTerm, Alacritty, Konsole, GNOME Terminal atau yang lainnya. Apabila tidak memungkinkan, maka Vim dapat diakses melalui GUI, seperti menggunakan gVim atau MacVim.

Vim versi GUI merupakan aplikasi yang mandiri (*standalone application*) – tidak membutuhkan *terminal* untuk jalan. Selain itu, perbedaan mendasar antara Vim versi GUI dan CLI adalah adanya *menubar* di dalam Vim GUI, *menubar* tersebut berisi koleksi perintah-perintah yang memungkinkan kamu untuk berinteraksi dengan Vim melalui *mouse*. Di luar dari kedua hal tadi, cara mengoperasikan Vim versi GUI sama saja seperti di CLI.

Esensi menggunakan Vim adalah efisiensi *keyboard*, jadi lebih baik membiasakan menggunakan Vim di dalam CLI, ketimbang harus memaksakan menggunakan Vim tapi dalam GUI. Bila memang lebih suka dengan GUI, lebih direkomendasikan menggunakan *non-modal editor* seperti VS Code atau semacamnya.

Tidak masalah juga bila menggunakan Vim versi GUI, karena di sistem operasi seperti Windows cukup sulit untuk memasang Vim kecuali menggunakan gVim. Kabar baiknya mulai dari versi 10, Windows memungkinkan kita untuk memasang Linux di dalam Windows, hal ini bernama WSL. Dengan WSL kita dapat memiliki pengalaman penuh menggunakan Linux di dalam Windows tanpa harus melakukan teknik *dual boot*. Karena WSL berjalan di dalam lingkungan Linux sendiri, sehingga kita dapat memasang Vim dengan mudah selayaknya pengguna Linux pada umumnya.

---

<sup>4</sup> "Terminal Emulator" [https://en.wikipedia.org/wiki/Terminal\\_emulator](https://en.wikipedia.org/wiki/Terminal_emulator)

Saat sudah berada di dalam Vim, perbedaan antara versi GUI dan CLI tidak begitu *kentara*, karena setiap perintah Vim sama-sama akan dieksekusi melalui *keyboard* – GUI dan CLI pada akhirnya hanya sebagai perantara saja.

## Sejarah Singkat

Vim terlahir karena adanya editor teks lain bernama Vi (penyebutannya dipisah: v-i dalam Bahasa Inggris) yang dibuat oleh Bill Joy<sup>5</sup> dan dirilis pertama kali tahun 1976. Awalnya editor teks ini bernama ex sampai tahun 1979 versi kedua dari ex rilis di bawah nama vi kependekan dari *visual*. Dan nama itulah yang sampai saat ini kita kenal.

Kemudian seseorang bernama Bram Moolenaar<sup>6</sup> mem-*fork* sumber kode dari Vi dan melakukan beberapa improvisasi, seperti menambah beberapa fitur yang tidak ada di dalam Vi sebelumnya.

Lalu pada tahun 1991 editor teks Vi yang sudah diimprovisasi dirilis ke publik oleh Bram dan diberi nama Vim kependekan dari **\*Vi IMPROVED\***.

Sederhananya, Vim adalah editor teks yang kodenya berbasis dari Vi dan dilakukan beberapa improvisasi ke dalamnya, seperti *syntax highlighter*, *multi-level undo*, menambahkan fitur *macros*, dan masih banyak lagi. Vim berlisensi GPL-compatible.

Selain Vim, ada juga editor teks bernama Neovim. Seperti Vim, Neovim adalah versi yang sudah diimproviasi (lagi) dari Vim. Bila diurutkan seperti ini jadinya:

---

<sup>5</sup> "Bill Joy" [https://en.wikipedia.org/wiki/Bill\\_Joy](https://en.wikipedia.org/wiki/Bill_Joy)

<sup>6</sup> "Bram Moolenaar" [https://en.wikipedia.org/wiki/Bram\\_Moolenaar](https://en.wikipedia.org/wiki/Bram_Moolenaar)



Motivasi terciptanya Neovim<sup>7</sup> ini adalah basis kode Vim yang terlalu “menyeramkan” untuk disentuh dan komunitas Vim yang tidak begitu cepat dalam melakukan pengembangan, karena hanya satu orang yang bertanggung jawab atas segala perubahan pada kode Vim, yaitu Bram Moolenaar.

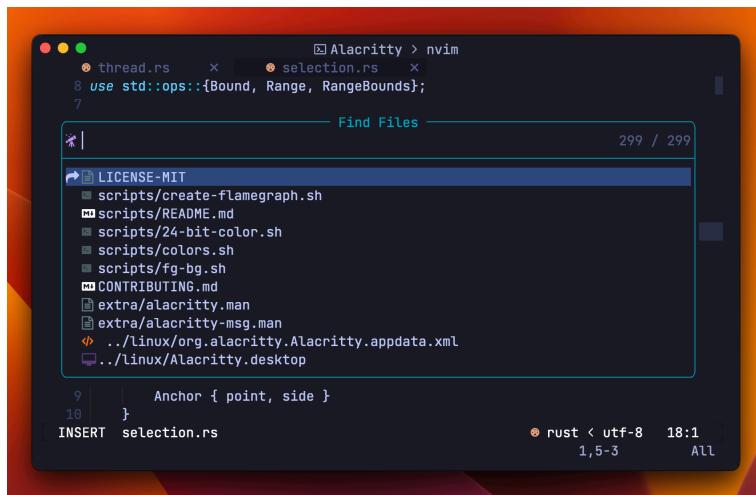
Walaupun arsitektur Vim sudah diubah di dalam Neovim, ekosistem *plugin* Vim dan Neovim tidak berbeda jauh. Karena banyak *plugin-plugin* dari Vim dapat dipasang di Neovim. Dan keduanya memiliki *keybinding* yang serupa.

Vim ataupun Neovim, keduanya sama-sama *extensible* dan *customizable*. Jadi keduanya bisa dipasangkan *plugin* pihak ketiga ataupun dikustomisasi untuk menyesuaikan kebutuhan kita.

Saya pribadi saat ini menggunakan Neovim, setelah sebelumnya menggunakan Vim. Karena saya merasa Neovim memiliki *plugin-plugin* dan fitur menarik untuk dicoba, dan salah satu yang membuat saya pindah ke Neovim adalah prihal *floating window*.

---

<sup>7</sup> "Introduction" <https://github.com/neovim/neovim/wiki/Introduction>



Neovim floating window

Kamu lihat kotak yang sedang melayang di atas? Ya, itu maksudnya! Mungkin kamu akan berpikir bahwa hal itu biasa saja, tapi bila dilakukan di dalam CLI maka butuh *effort* untuk melakukannya dan terlihat lebih keren.

Dan, ya, saya mencoba mem-VS-code-kan Neovim saya di dalam CLI!

## Neovim

Cara mengoperasikan Vim dan Neovim sama saja. Katakanlah kamu sudah ahli menggunakan Vim selama bertahun-tahun dan tiba-tiba beralih ke Neovim, maka perintah-perintah umum yang biasa kamu gunakan di Vim, juga akan bekerja di dalam Neovim – tanpa harus belajar lagi dari awal.

Dalam hal arsitektur Neovim dirancang agar lebih berkinerja dan dapat dipelihara. Ini juga yang menjadi alasan mengapa Neovim menggunakan Lua ketimbang bahasa yang lebih berat seperti JavaScript atau Python. Basis kode Neovim yang dapat dipelihara menghasilkan komunitas yang lebih cepat berkembang.

Selain itu, konfigurasi Vim dapat kita gunakan di dalam lingkup Neovim. Sebagai tambahan, kita juga dapat menulis konfigurasi Vim dengan bahasa Lua. Saya pribadi menulis konfigurasi Neovim dengan Lua karena lebih rapih dan terorganisasi dengan baik secara subjektif. *Plugin-plugin* Neovim yang ditulis dengan bahasa Lua juga banyak.

Neovim mendukung LSP (Language Server Protocol) secara bawaan. LSP memungkinkan Neovim berkomunikasi dengan *server* bahasa pemrograman untuk mengaktifkan fitur seperti *code completion*, *documentation*, *code formatting* atau *go-to-definition*.

Pada buku ini kita akan belajar menggunakan Vim, setelahnya kita akan menggunakan Neovim dan mengkustomisasinya agar dapat digunakan sebagai editor teks sehari-hari, bahkan dapat menjadi IDE (Integrated Development Environment) andalan kita untuk bekerja.

Mungkin kamu ingin bertanya mengapa kita tidak langsung menggunakan Neovim saja, sehingga tidak perlu pindah-pindah. Hal ini dilakukan agar memberi pemahaman bahwa pada dasarnya kedua program Vim dan Neovim tidak banyak berbeda, kecuali dari sisi arsitektur seperti yang disebutkan sebelumnya.

## Konfigurasi

Ketika membuka Vim pertama kali, tidak perlu berekspektasi banyak hal, seperti terdapat *syntax highlighter*, *line number*, apalagi *code completion*. Semua itu tidak ada. Vim maupun Neovim adalah editor teks yang *unopinionated*, yang berarti mereka tidak punya pandangan apapun terhadap bagaimana kita menggunakan editor teks.

Bila kita menginginkan *syntax highlighter* pada kode yang kita tulis, maka kita perlu mengaktifkannya melalui konfigurasi, begitu juga dengan fitur lain seperti *line number*. Untuk fitur-fitur yang lebih lanjut seperti *file explorer*, *code formatting*, *code completion*, atau *code linting* kita perlu mengaturnya secara mandiri.

Memang benar LSP sudah tersedia secara bawaan pada Neovim, tapi ia hidup di dalam repositori yang berbeda, sehingga kita perlu memasangnya dan melakukan konfigurasi secara mandiri hingga dapat menggunakan fitur-fitur seperti *completion*, *documentation*, *goto-definition*, *format* hingga *diagnostic*.

Terlihat lebih repot ketimbang editor teks seperti VS Code yang pada dasarnya semuanya sudah ada tanpa harus kita konfigurasi secara mandiri. Namun hal ini justru yang membuat saya pribadi semakin tertarik dengan ekosistem Vim. Saya merasa lebih bebas dan merasa keren ketika menulis konfigurasi yang membuat Vim maupun Neovim saya terlihat cantik dan dapat diandalkan.

Pada dasarnya editor teks seperti VS Code juga berdasarkan konfigurasi, namun bedanya semua konfigurasi itu sudah ada secara bawaan. Kita tidak perlu melakukannya secara mandiri, kecuali bila kita ingin mengkustomisasinya – kita dapat mengubah konfigurasi bawaannya. Ini yang membuat editor teks seperti VS Code terlihat lebih sederhana untuk digunakan dibanding dengan Vim.

Menulis konfigurasi Vim secara mandiri kedengarannya merepotkan, tapi faktanya memang seperti itu. Kendati merepotkan, kita tidak melulu menulis konfigurasi Vim keseluruhan dari awal. Umumnya pengguna Vim menggunakan konfigurasi dari sesama pengguna Vim lain sebagai *starting point* konfigurasinya, setelah itu disesuaikan lagi.

Dengan seperti ini kita memiliki kendali penuh atas editor teks yang kita gunakan, kita dapat menyesuaikan fitur apa saja yang ada di dalam Vim.

Banyak konfigurasi Vim yang dapat kita gunakan dan tersebar di internet, mulai dari yang minimal hingga yang kaya akan fitur seperti AstroNvim. Tidak seperti Neovim, AstroNvim bukanlah *program* terusan dari Vim, melainkan sebuah *layer* IDE untuk Neovim yang berisi koleksi konfigurasi Neovim agar dapat dijadikan sebagai IDE.

AstroNvim menyediakan banyak fitur-fitur dasar yang dibutuhkan, seperti *code completion*, *file explorer*, *fuzzy finder*, *linting*, *formatting*, *debugging*, dan yang terpenting adalah kecepatan. Kedengarannya memang menjadi *opinionated*, tapi AstroNvim memungkinkan kita untuk mengkustomisasi konfigurasi bawaan mereka – kendali penuh masih ada di tangan kita. Lagipula hidup terlalu pendek untuk menulis konfigurasi Neovim dari awal.

## Case Sensitive

Perintah-perintah pada Vim bersifat *case-sensitive*. Sebagai contoh, perintah d dengan D itu berbeda. Itu berarti apabila tombol **CAPSLOCK keyboard** kamu menyala dan kamu menekan tombol d, maka kamu menjalankan perintah D. Karena menekan tombol d pada saat CAPSLOCK menyala akan menghasilkan D.

Begitu juga dengan perintah-perintah yang lain yang saya akan jelaskan di bagian-bagian berikutnya. Hal ini saya sampaikan di awal agar tidak terjadi salah paham dan membungkungkan di kemudian waktu.

# Memasang Vim

Vim tersedia secara luas di berbagai sistem operasi. Bila kamu menggunakan sistem operasi berbasis Unix atau Unix-*like* seperti macOS atau Linux, maka umumnya sudah terpasang Vim di dalamnya.

Bila kamu menggunakan sistem operasi seperti Windows, maka kamu bisa menggunakan gVim atau bila kamu menggunakan Git Bash, di dalamnya juga tersedia Vim. Tapi bila kamu menggunakan Windows 10 atau di atasnya, saya lebih menyarankan untuk menggunakan WSL (Windows Subsystem Linux) untuk mendapatkan pengalaman Linux yang lebih baik.

Pada buku ini kita akan menggunakan Vim di dalam Terminal jika memungkinkan, dan menggunakan versi GUI bila tidak memungkinkan.

Membuat langkah-langkah yang rinci untuk sistem operasi Unix-*like* seperti Linux akan terlalu luas jangkauannya, mengingat Linux memiliki banyak sekali distribusi yang berbeda-beda. Masing-masing distro memiliki *package manager* yang berbeda, sehingga akan membuat langkah-langkahnya pun berbeda dari yang lain.

Di buku ini, saya hanya mencontohkan memasang Vim pada distribusi Ubuntu. Untuk distribusi yang lain seharusnya langkah-langkahnya tidak berbeda jauh. Untuk itu saya menyarankan untuk pergi ke halaman unduh Vim<sup>8</sup> untuk informasi yang lebih lengkap.

## macOS

Biasanya Vim sudah tersedia di banyak versi macOS, untuk memverifikasi ketersediaan Vim, kita dapat membuka program Terminal dan mengetik perintah `vim`.

---

<sup>8</sup> "Unduh Vim" <https://www.vim.org/download.php>

```
mhdnaualazhar -- zsh -- 112x33
: vim --version
VIM - Vi IMproved 9.0 (2022 Jun 28, compiled Sep 30 2022 03:10:57)
macOS version - arm64
Included patches: 1-270
Compiled by root@apple.com
Normal version without GUI. Features included (+) or not (-):
+acl           +file_in_path     -mouse_urxvt      -tag_any_white
+arabic         +find_in_path    -mouse_xterm       -tcl
+autocmd        +float           -multi_byte       -termguicolors
+autochdir      +folding          -multi_lang        +terminal
+autoencodername +fotter          -multi_theme      +terminfo
+ballooon_eval   +fork()          -netbeans_intg   +termresponse
+ballooon_eval_term -gettext         -num64           +textobjects
+browse          -hangul_input    +packages          +textxprop
+builtin_terms   +iconv           +path_extra       +timers
+byte_offset     +insert_expand   -perl             +title
+channel         +ipv6            +persistent_undo -toolbar
+cindent         +job              +popupwin        +user_commands
-clientserver    +jumplist        +postscript       -varTabs
+clipboard       -lisp            +profile          +win32script
+column_compl   +lambda          -python           +wininfo
+cmdline_hist   -langmap         +python3          +virtualedit
+cmdline_info   +libcall          -quickfix        +visual
+comments        +linebreak        +spell            +visualextra
+conceal         +lispindent       -reltime         +vreplace
+cryptv          +listcmds        -rightleft       +wildignore
+cscope          +localmap        -ruby             +wildmenu
+cursorbind      -lua              +scrollbind      +windows
+cursorshape     +menu            +signs            +writebackup
+diff            +mksession       +spelldict       -X11
+digraphs        +modify_fname   -sodium           -xfontset
+dtm             +mouse           -sound            -xim
```

### Memeriksa ketersediaan program Vim

Bila Terminal menampilkan hasil kira-kira seperti di atas, Vim berarti tersedia. Terminal mungkin menampilkan hasil yang lain seperti mengindikasikan bahwa perintah `vim` tidak dapat ditemukan, itu berarti Vim tidak tersedia. Jika demikian, kita dapat memasang Vim di mesin macOS melalui Homebrew<sup>9</sup> dengan tahap-tahap berikut:

1. Tetap berada di dalam *program* Terminal
2. Jalankan perintah `xcode-select --install` untuk memasang alat-alat *command-line* di macOS
3. Untuk memasang Homebrew gunakan perintah `/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"`
4. Dalam kasus tertentu, mungkin kamu perlu menambahkan alamat *bin* Homebrew secara manual ke dalam berkas konfigurasi Shell kamu, misal `~/.zprofile` bila kamu menggunakan Zsh

<sup>9</sup> "Sebuah package manager untuk macOS atau Linux" <https://brew.sh>

```

Warning: /opt/homebrew/bin is not in your PATH.
  Instructions on how to configure your shell for Homebrew
  can be found in the 'Next steps' section below.
==> Installation successful!

==> Homebrew has enabled anonymous aggregate formulae and cask analytics.
Read the analytics documentation (and how to opt-out) here:
https://docs.brew.sh/Analytics
No analytics data has been sent yet (nor will any be during this install run).

==> Homebrew is run entirely by unpaid volunteers. Please consider donating:
https://github.com/Homebrew/brew#donations

==> Next steps:
- Run these three commands in your terminal to add Homebrew to your PATH:
  echo '# Set PATH, MANPATH, etc., for Homebrew.' >> /Users/nuaval/.zprofile
  echo 'eval "$( /opt/homebrew/bin/brew shellenv )"' >> /Users/nuaval/.zprofile
  eval "$( /opt/homebrew/bin/brew shellenv )"

```

Instruksi untuk mendaftarkan Homebrew ke PATH

5. Sekarang kita dapat memasang Vim melalui Homebrew dengan perintah `brew install vim`

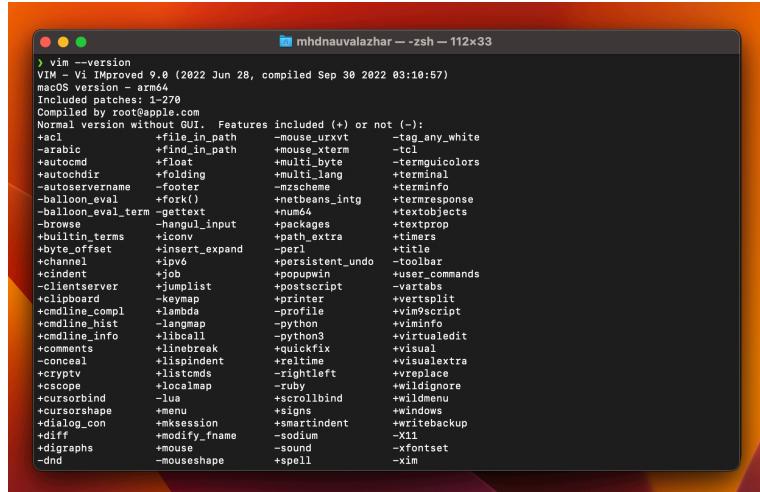
```

mhdnaualzhar -- zsh -- 112x33
> brew install vim
Running `brew update --auto-update`...
==> Auto-updated Homebrew!
Updated 6 taps (heroku/brew, shivammathur/php, homebrew/cask-versions, homebrew/core, homebrew/cask and planetsec/ale/tap).
==> Formulae
birdgen               corrosion
==> New Casks
cad-assistant        element-nightly    quiet-reader      rapidapi
You have 61 outdated formulae and 3 outdated casks installed.
You can upgrade them with brew upgrade
or list them with brew outdated.
==> Downloading https://ghcr.io/v2/homebrew/core/lua/manifests/5.4.4.1
#####
==> Downloading https://ghcr.io/v2/homebrew/core/lua/blobs/sha256:c26ccf8003f231
#####
==> Downloading from https://pkg-containers.githubusercontent.com/ghcr1/blobs/sh
#####
==> Downloading https://ghcr.io/v2/homebrew/core/ncurses/manifests/6.3
#####
==> Downloading https://ghcr.io/v2/homebrew/core/ncurses/blobs/sha256:b534276b85
#####
==> Downloading from https://pkg-containers.githubusercontent.com/ghcr1/blobs/sh
#####
==> Downloading https://ghcr.io/v2/homebrew/core/openbsd/manifests/1.1
#####
==> Downloading https://ghcr.io/v2/homebrew/core/openbsd/blobs/sha256:3a7812
#####
==> Downloading from https://pkg-containers.githubusercontent.com/ghcr1/blobs/sh
#####
==> Downloading https://ghcr.io/v2/homebrew/core/berkeley-db/manifests/18.1.48.1
#####
==> Downloading https://ghcr.io/v2/homebrew/core/berkeley-db/blobs/sha256:9f95fa
#####
==> Downloading from https://pkg-containers.githubusercontent.com/ghcr1/blobs/sh

```

Memasang Vim melalui Homebrew

6. Tunggu proses instalasi hingga selesai. Ketik kembali perintah `vim --version` untuk memverifikasi program Vim sudah terpasang



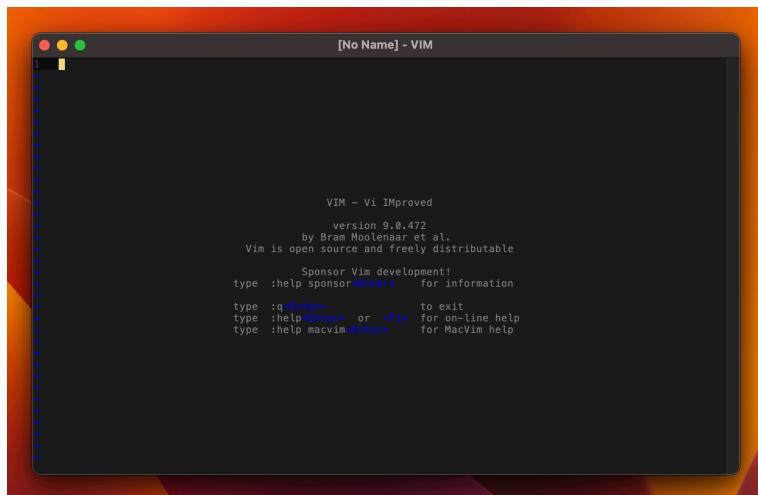
```
mhdnaualazhar -- zsh -- 112x33
: vim --version
VIM - Vi IMproved 9.0 (2022 Jun 28, compiled Sep 30 2022 03:10:57)
macOS version - arm64
Included patches: 1-270
Compiled by root@apple.com
Normal version without GUI. Features included (+) or not (-):
+acl           +file_in_path    -mouse_urxvt      -tag_any_white
+arabic         +find_in_path   -mouse_xterm      -tcl
+autocmd        +float          -multi_byte      -termguicolors
+autochdir      +folding        -multi_lang       +terminal
+autochservername -fzf            -oschrome        +terminal2
+ballooon_eval  +fork()        -osbeans_intg     +textresponse
+ballooon_eval_term -gettext       -num64          +textobjects
+browse         -hangul_input   +packages         +textprop
+builtin_terms  +iconv          +path_extra      +timers
+byte_offset    +insert_expand  -perl             +title
+channel        +ipv6           +persistent_undo -toolbar
+cindent         +job             +popupwin       +user_commands
-clientserver   +jumplist       +postscript      -vartabs
+clipboard       -lisp            +profile         +virtualedit
+clipbard       +lispd8          +python          +viminfo
+cmdline_compl  +langmap        +python3         +visual
+cmdline_hist   +libcall        +quickfix       +visualextra
+cmdline_info   +linebreak      +reltime         +vreplace
+comments       +listcmds       -rightleft      +wildignore
+cryptv         +localmap       -ruby            +wildmenu
+cscope          +lua             +scrollbind     +windows
+cursorbind     +mu             +signs          +writebackup
+cursorshape    +mzscheme       +sparendent     -X11
+diff           +modify_fname  -sodium          -xfontset
+digraphs       +mouse          -sound          -xim
-dnd            -mouseshape     +spell           -xim
```

Memeriksa ketersediaan Vim setelah dipasang

Bagian "9.0" pada hasil perintah `vim --version` adalah versi Vim yang saat ini terpasang. Mungkin kamu memiliki versi yang berbeda saat membaca buku ini, seperti "9.1", "9.2", dan seterusnya – itu tidak masalah.

Selain melalui Homebrew, kita dapat menggunakan MacVim untuk versi GUI dari Vim di macOS, tapi saya tidak merekomendasikan program tersebut, sebab kita sudah dapat memasang Vim di dalam Terminal.

Barangkali kamu penasaran, tampilan MacVim seperti ini:



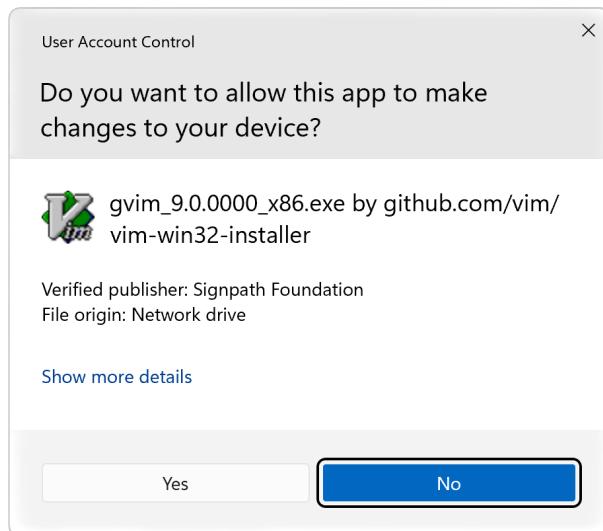
Tampilan MacVim

MacVim tidak membutuhkan Terminal untuk dapat dijalankan, itulah salah satu perbedaan antara Vim versi *terminal* dan versi GUI. Sama halnya dengan Vim versi GUI di sistem operasi lain seperti Windows.

## Windows

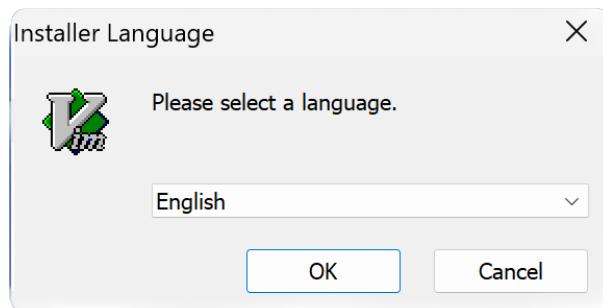
Bila kamu menggunakan Windows 10, sangat disarankan untuk menggunakan WSL. Namun, seandainya kamu tidak tertarik dengan Linux dan enggan memasang WSL agar dapat menggunakan Vim, kamu dapat memasang program bernama gVim di Windows.

1. Pergi ke halaman rilis *installer* gVim:  
(<https://github.com/vim/vim-win32-installer/releases>)
2. Unduh gVim versi paling baru (saat ini versi 9.0)
3. Klik ganda pada berkas *exe* yang baru saja diunduh, biasanya akan muncul *dialog User Account Control* di Windows, pilih *Yes*



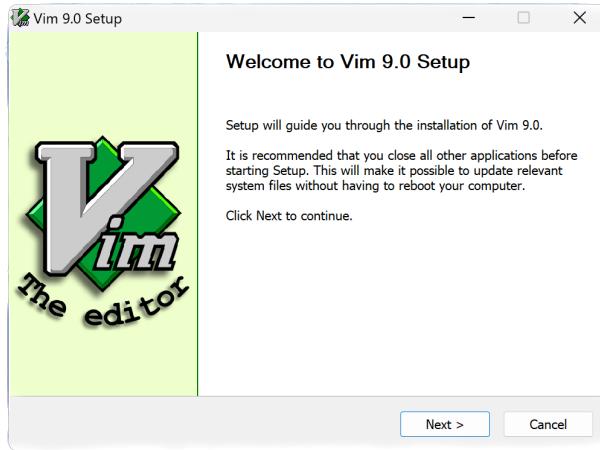
Dialog User Account Control di Windows

4. Pilih bahasa untuk *installer* dan klik tombol *OK*



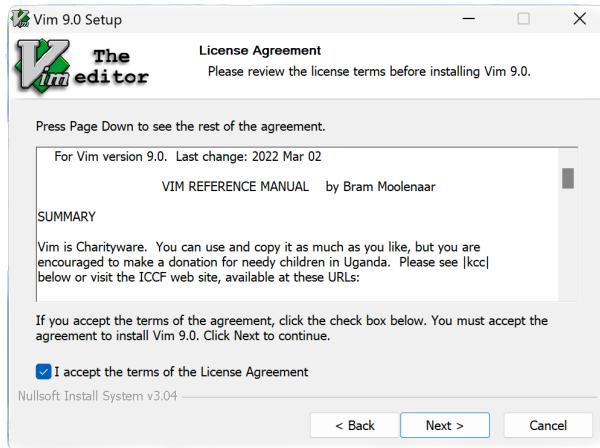
Pilih bahasa

5. Berikutnya akan muncul tampilan *Welcome* dari *installer* gVim, klik tombol *Next >* saja



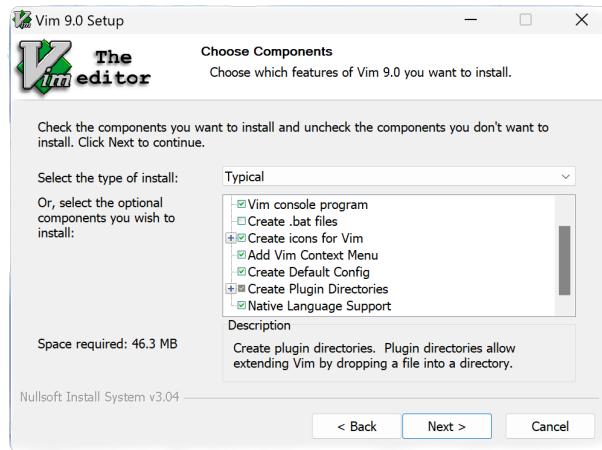
Welcome screen installer

6. Untuk memasang gVim, kamu harus setuju dengan *License Agreement* yang dibuat, maka dari itu klik kotak cek bertuliskan "I accept ..." dan klik tombol *Next >* untuk melanjutkan



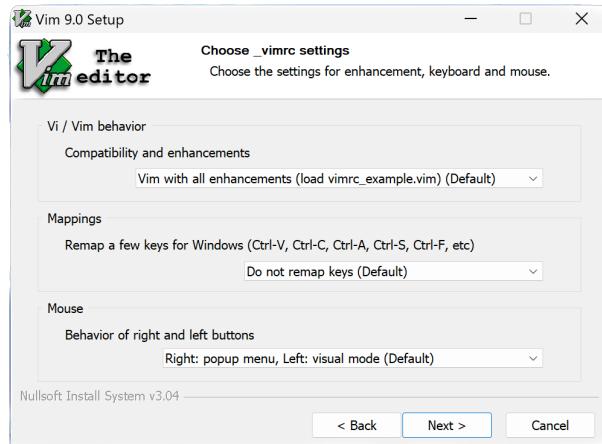
License Agreement

7. Bagian ini kita dapat memiliki fitur apa saja yang ingin dipasang, biarkan saja dengan opsi bawaan dan klik tombol *Next >*



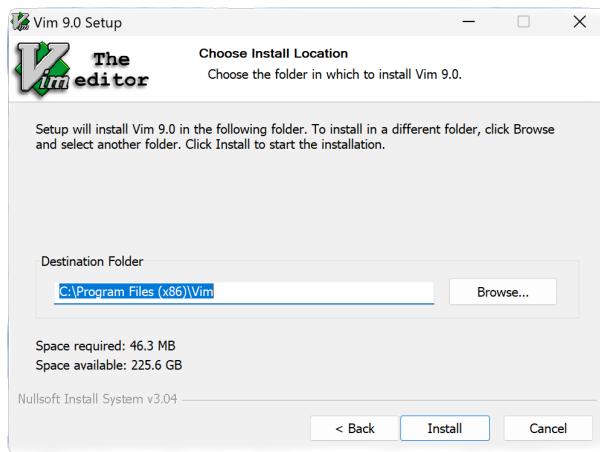
#### Memilih fitur-fitur gVim

8. Selain itu, kita juga dapat memiliki pengaturan untuk peningkatan fungsionalitas Vim, di bagian ini kita dapat abaikan agar menggunakan opsi bawaan gVim, klik tombol *Next >*



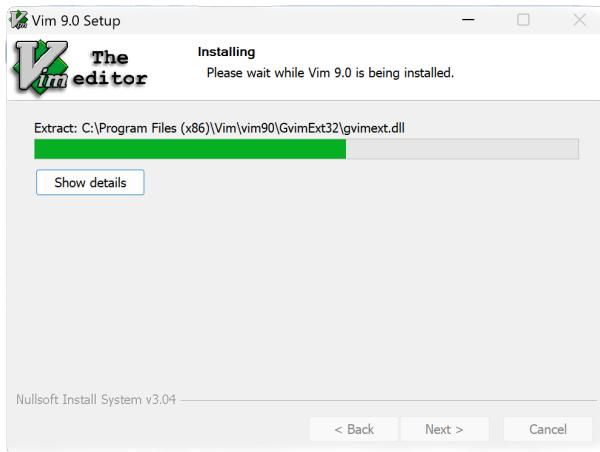
#### Konfigurasi tambahan gVim

9. Pilih lokasi pemasangan program gVim, biasanya di **C:\Program Files (x86)\Vim**



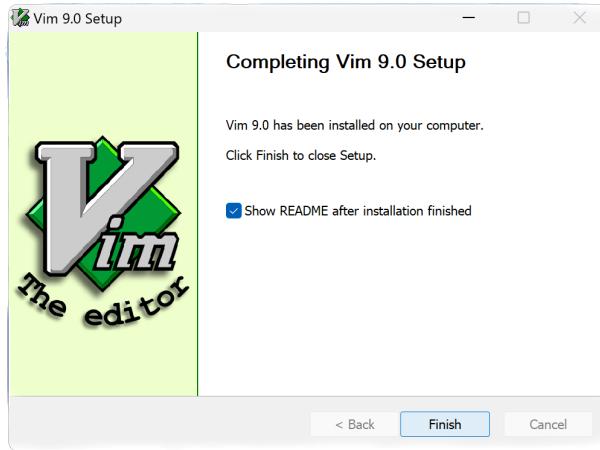
Lokasi pemasangan gVim

## 10. Tunggu proses pemasangan hingga selesai



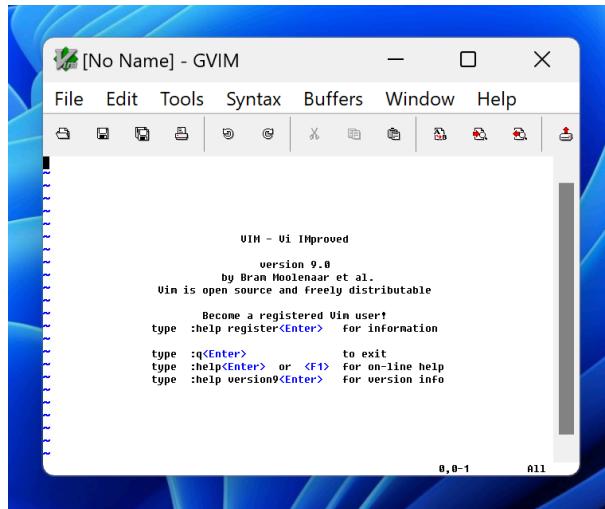
Proses pemasangan gVim

## 11. Klik tombol *Finish* untuk menutup program *installer*



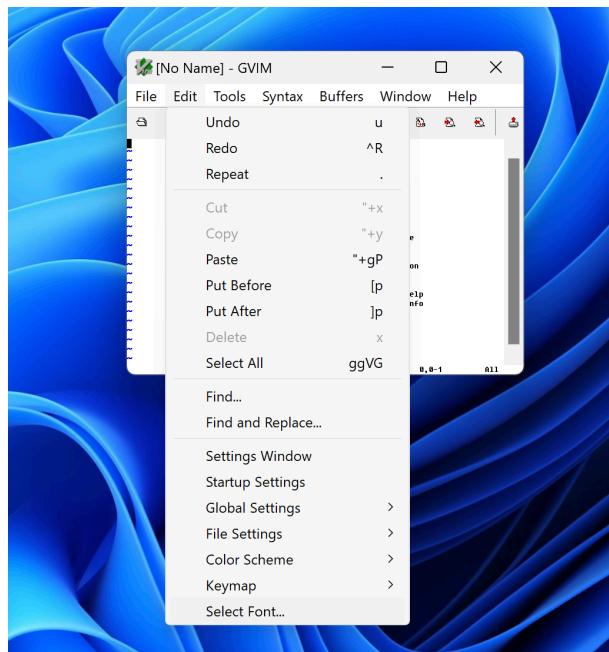
### Pemasangan selesai

Buka program gVim yang baru saja dipasang, tampilannya kira-kira seperti ini:



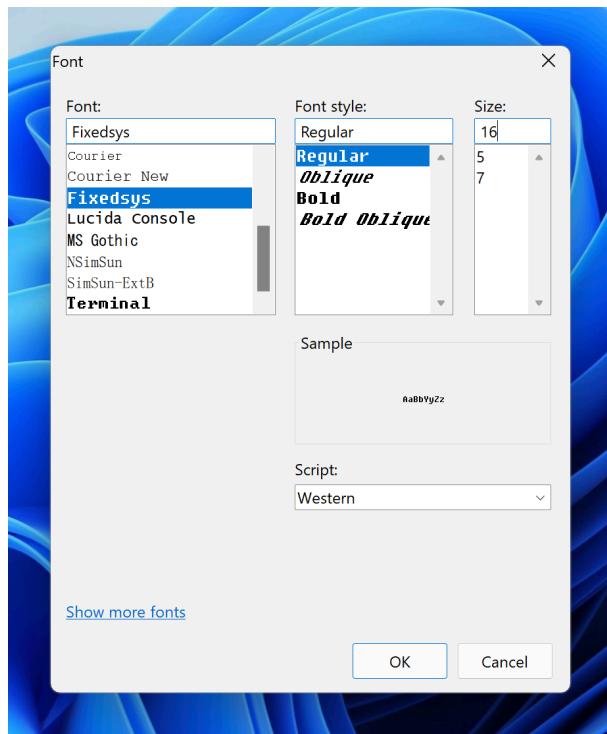
Tampilan Program gVim

Pada kasus saya, gVim memiliki ukuran huruf yang sangat kecil. Untuk mengubah ukuran huruf kita dapat pergi ke menu *Edit > Select Font*



Memilih menu Select Font

Maka akan terbuka jendela baru untuk pengaturan jenis, gaya dan ukuran huruf. Saya menggunakan ukuran 16 agar tulisan di gVim dapat terlihat lebih jelas.



Jendela pengaturan huruf gVim

Sekarang seharusnya gVim sudah memiliki ukuran huruf yang lebih besar.

## Windows Subsystem Linux

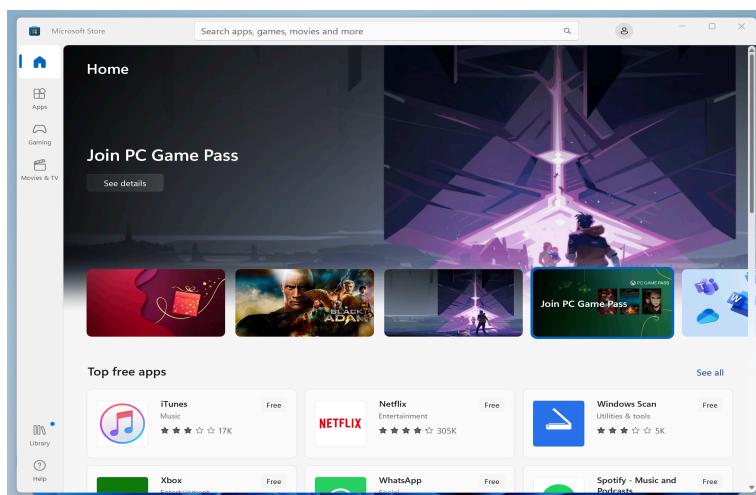
Mulai dari Windows 10 dan di atasnya, Microsoft memungkinkan kita untuk memasang distribusi Linux di dalam Windows secara langsung tanpa harus melalui metode tradisional seperti menggunakan *virtual machine* atau *dualboot setup*. Dengan seperti itu, kita dapat menikmati lingkungan Windows dan Linux secara bersamaan.

Saya pribadi menyarankan menggunakan Vim di dalam lingkungan Linux melalui WSL, karena selain Vim sudah ada secara bawaan di banyak distro Linux, juga kita dapat mengakses aplikasi, *utilities*, dan *command-line tools* yang tersedia di Linux.

Terdapat beberapa distribusi Linux yang tersedia di WSL, seperti Ubuntu, Debian, OpenSUSE, Kali, Debian, Arch Linux, dan masih banyak lagi. Pada langkah-langkah berikut, saya menggunakan Windows 11 dan akan memilih Ubuntu, tidak masalah seandainya kamu hendak memilih distribusi Linux yang lain, seharusnya langkah-langkahnya tidak jauh berbeda.

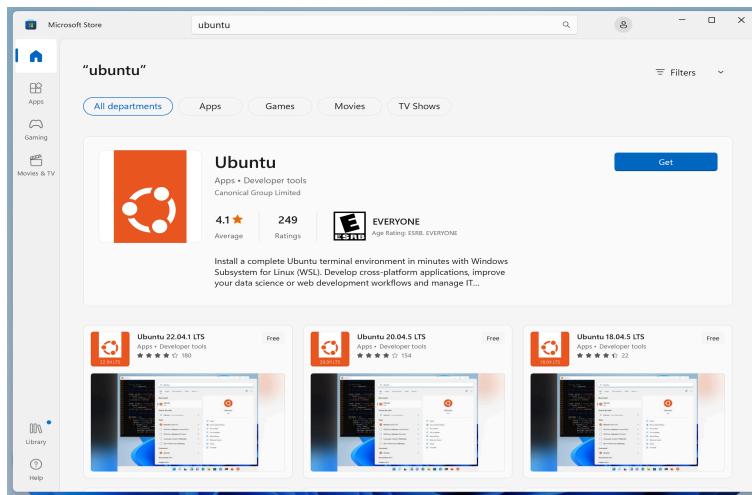
Untuk memasang WSL di Windows, berikut langkah-langkahnya:

1. Buka aplikasi Microsoft Store



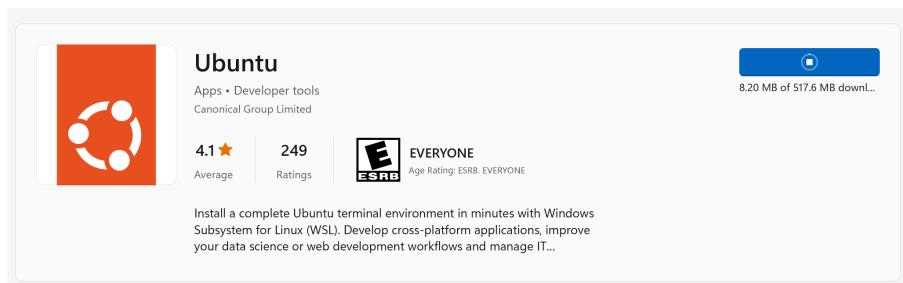
Aplikasi Microsoft Store

2. Ketik "Ubuntu" di kolom pencarian



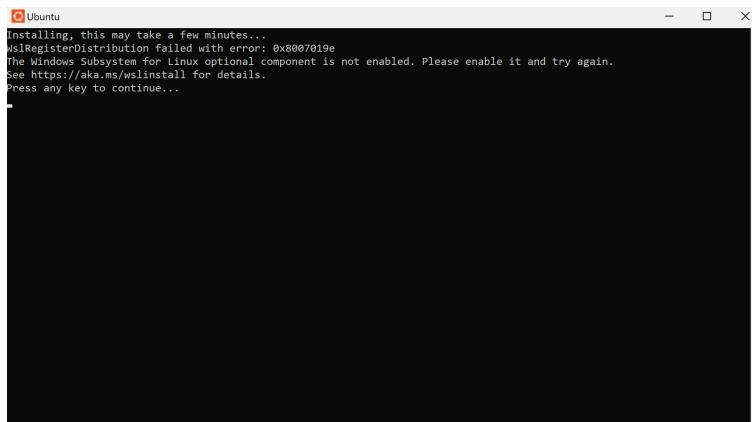
Mencari aplikasi Ubuntu

3. Klik tombol "Get" berwarna biru untuk mulai memasangnya dan tunggu proses unduhan hingga selesai



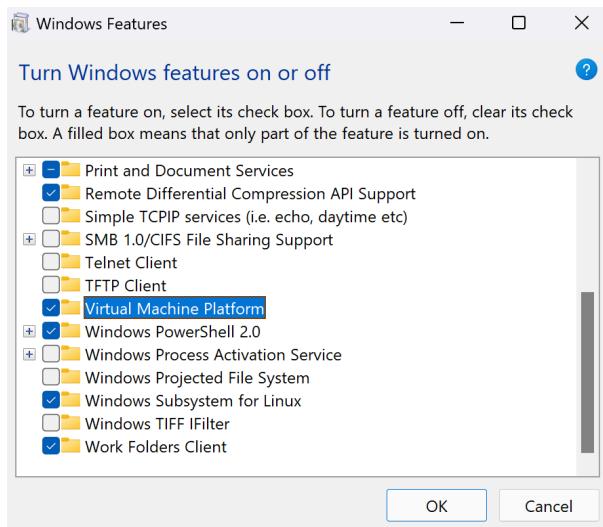
Mengunduh dan memasang Ubuntu

4. Jika sudah selesai, buka aplikasi tersebut dan akan muncul jendel baru yang menunjukkan bahwa ada fitur yang belum diaktifkan untuk mulai menggunakan WSL



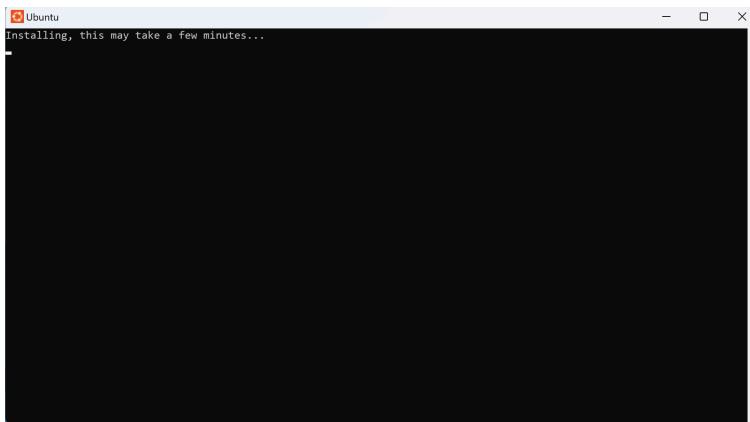
Jendela WSL Ubuntu

5. Buka aplikasi Control Panel, pergi ke *Programs* dan klik pada *Turn Windows features on or off*, berikan ceklis pada kedua fitur: *Virtual Machine Platform* dan *Windows Subsystem for Linux*



Mengaktifkan fitur yang dibutuhkan WSL

6. Buka kembali aplikasi Ubuntu, sekarang tunggu proses instalasi selesai



Memulai proses instalasi Ubuntu

7. Masukkan *username* dan *password* untuk instalasi Ubuntu kamu

```
mhdnaulazhar@MUHAMADNAUV941C:~  
Installing, this may take a few minutes...  
Please create a default UNIX user account. The username does not need to match your Windows username.  
For more information visit: https://aka.ms/wslusers  
Enter new UNIX username: mhdnaulazhar  
New password:  
Retype new password:  
passwd: password updated successfully  
Installation successful  
Windows Subsystem for Linux is now available in the Microsoft Store!  
You can upgrade by running 'wsl.exe --update' or by visiting https://aka.ms/wslstorepage  
Installing WSL from the Microsoft Store will give you the latest WSL updates, faster.  
For more information please visit https://aka.ms/wslstoreinfo  
To run a command as administrator (use "root"), use "sudo <command>".  
See "man sudo_root" for details.  
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 4.4.0-22621-Microsoft aarch64)  
 * Documentation: https://help.ubuntu.com  
 * Management: https://landscape.canonical.com  
 * Support: https://ubuntu.com/advantage  
This message is shown once a day. To disable it please create the  
/home/mhdnaulazhar/.hushlogin file.  
mhdnaulazhar@MUHAMADNAUV941C:~$
```

Instalasi WSL Ubuntu selesai

8. Verifikasi ketersediaan Vim di dalam WSL Ubuntu dengan perintah `vim --version`

```
mhdnaulazhar@MUHAMADNAUV941C: ~
/mhnaulazhar@MUHAMADNAUV941C:~$ vim --version
VIM - VI IMproved 8.2 (2019 Dec 12, compiled Sep 13 2022 09:35:02)
Included patches: 1-3995, 4563, 4646, 4774, 4895, 4899, 4901, 4919
Modified by team-vim@tracker.debian.org
Compiled by team-vim@tracker.debian.org
huge version without GUI. Features included (+) or not (-):
+acl          +file_in_path    +mouse_urxvt      -tag_any_white
+arabic       +find_in_path   +mouse_xterm      -tcl
+autocmd      +float          +multi_byte       +terminalguicols
+autochdir    +folding         +multi_lang        +terminalinfo
+autoservername +footer         +mzscheme         +termresponse
+balloon_eval  +fork()         +netbeans_intg   +textobjects
+balloon_eval_term +gettext        +num64           +textprop
+browse        +hangul_input   +packages          +textxprop
+builtin_terms +iconv          +path_extra       +timers
+byte_offset   +insert_expand  -perl             +title
+byte_offset   +insert_expand  +persistent_undo +toolbar
+changelog     +ipv6           +popupwin        +user_commands
+cindent       +jbig2decode   +postscript       +varargs
+clientserver  +jumplist       +printer          +verysplit
+clipboard     +keymap         +profile          +vimscript
+cmdline_compl +lambda         +python           +viminfo
+cmdline_hist  +langmap        +python3          +virtualedit
+cmdline_info  +libcall        +quickfix        +visual
+comments      +linebreak      +reltime         +visualextra
+conceal       +lispindent     +rightleft      +replace
+cryptv       +listcmds      -ruby            +wildignore
+cscope        +localmap      +scrollbind     +wildmenu
+cursorbind    +lua            +signs           +windows
+cursorshape   +menu          +smartindent     +writebackup
+mksession
```

Memeriksa ketersediaan Vim di WSL Ubuntu

## 9. Selesai

Kamu dapat menggunakan aplikasi Ubuntu untuk mengakses WSL Ubuntu di dalam Windows, atau menutupnya dan menggunakan media lain seperti PowerShell atau Command Prompt.

Untuk menjalankan WSL di Command Prompt dapat menggunakan perintah `wsl`.

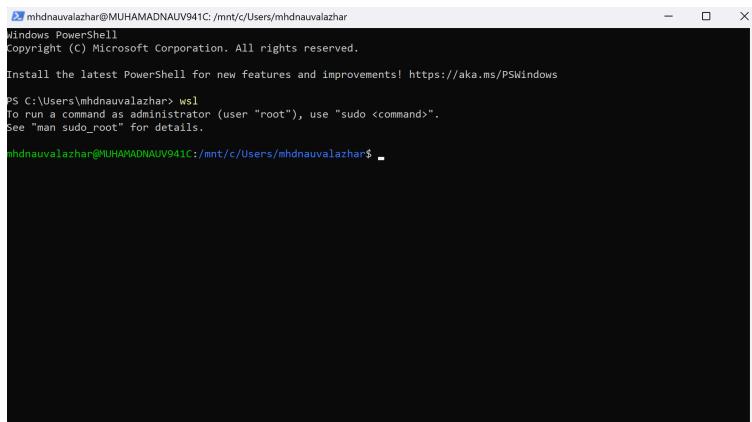
```
mhdnaulazhar@MUHAMADNAUV941C:/mnt/c/Users/mhdnaulazhar
Microsoft Windows [Version 10.0.22621.525]
(c) Microsoft Corporation. All rights reserved.

C:\Users\mhdnaulazhar>wsl
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

mhdnaulazhar@MUHAMADNAUV941C:/mnt/c/Users/mhdnaulazhar$
```

WSL di Command Prompt

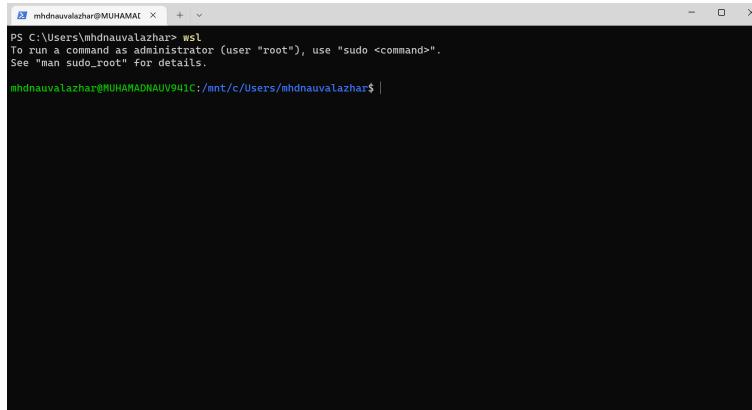
Untuk PowerShell juga sama seperti sebelumnya.



```
mhdnaualzhar@MUHAMADNAUV941C:/mnt/c/Users/mhdnaualzhar  
Windows PowerShell  
Copyright (C) Microsoft Corporation. All rights reserved.  
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows  
PS C:\Users\mhdnaualzhar> wsl  
To run a command as administrator (user "root"), use "sudo <command>".  
See "man sudo_root" for details.  
mhdnaualzhar@MUHAMADNAUV941C:/mnt/c/Users/mhdnaualzhar$ -
```

WSL di PowerShell

Selain itu, kamu juga dapat menggunakan aplikasi Terminal di Windows untuk mendapatkan pengalaman CLI yang lebih baik.



```
mhdnaualzhar@MUHAMADNAUV941C: | + ~  
PS C:\Users\mhdnaualzhar> wsl  
To run a command as administrator (user "root"), use "sudo <command>".  
See "man sudo_root" for details.  
mhdnaualzhar@MUHAMADNAUV941C:/mnt/c/Users/mhdnaualzhar$ |
```

WSL di dalam PowerShell dengan aplikasi Terminal

Ketika berada di dalam WSL, sama saja seperti berada di dalam Linux pada umumnya, kamu dapat mengakses *command-line tools* seperti `ls`, `mkdir`, `curl`, dan masih banyak lagi. Pada bagian-bagian berikutnya di buku ini, kamu dapat mengikuti panduan untuk bagian Linux, ketika saya menyebut Windows itu berarti saya merujuk pada gVim bukan WSL.

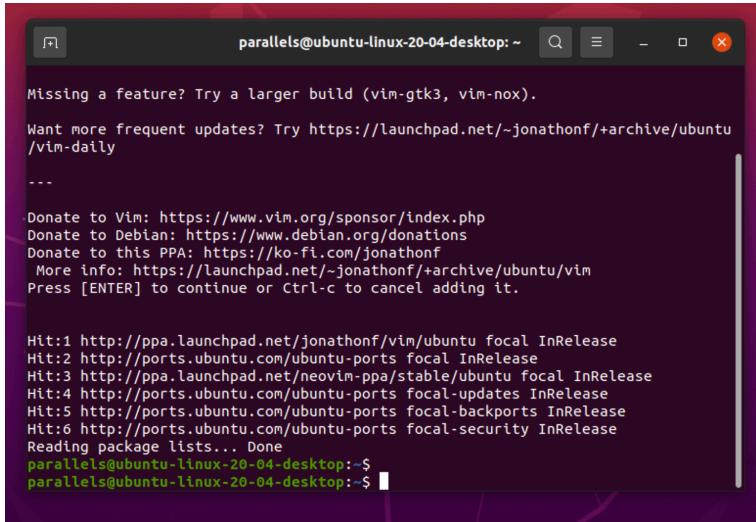
# Unix-like

Seperti yang sudah dijelaskan sebelumnya, saya hanya membuat langkah-langkah untuk Linux Ubuntu saja. Untuk sistem operasi atau distribusi Linux jenis lain, dapat mengikuti langkah-langkah berikut sebagai referensi namun dengan caranya masing-masing.

Lagipula mengajari pengguna sistem operasi Unix-like untuk memasang Vim seperti mengajari para pecinta alam untuk buang sampah sembarangan.

Berikut ini langkah-langkah untuk memasang Vim di Ubuntu:

1. Buka *program* Terminal
2. Untuk mendapatkan informasi paket Vim, jalankan perintah  
`sudo add-apt-repository ppa:jonathonf/vim`

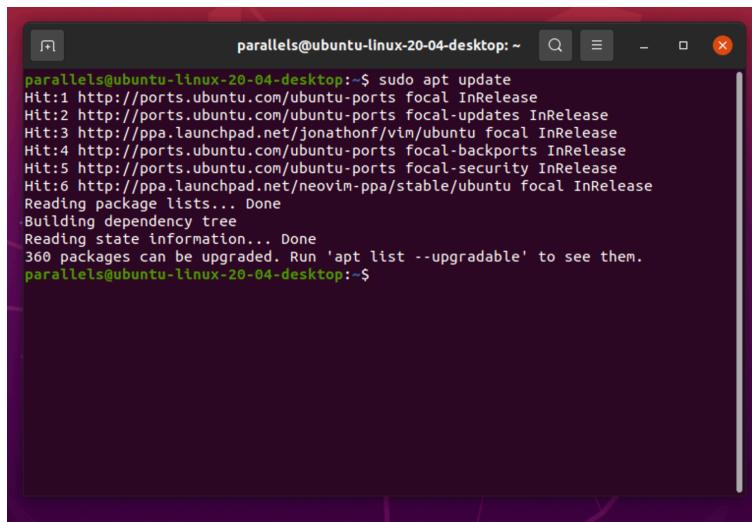


```
parallels@ubuntu-linux-20-04-desktop: ~
Missing a feature? Try a larger build (vim-gtk3, vim-nox).
Want more frequent updates? Try https://launchpad.net/~jonathonf/+archive/ubuntu/vim-daily
---
Donate to Vim: https://www.vim.org/sponsor/index.php
Donate to Debian: https://www.debian.org/donations
Donate to this PPA: https://ko-fi.com/jonathonf
More info: https://launchpad.net/~jonathonf/+archive/ubuntu/vim
Press [ENTER] to continue or Ctrl-c to cancel adding it.

Hit:1 http://ppa.launchpad.net/jonathonf/vim/ubuntu focal InRelease
Hit:2 http://ports.ubuntu.com/ubuntu-ports focal InRelease
Hit:3 http://ppa.launchpad.net/neovim-ppa/stable/ubuntu focal InRelease
Hit:4 http://ports.ubuntu.com/ubuntu-ports focal-updates InRelease
Hit:5 http://ports.ubuntu.com/ubuntu-ports focal-backports InRelease
Hit:6 http://ports.ubuntu.com/ubuntu-ports focal-security InRelease
Reading package lists... Done
parallels@ubuntu-linux-20-04-desktop: ~
parallels@ubuntu-linux-20-04-desktop: ~
```

Menambahkan informasi paket Vim

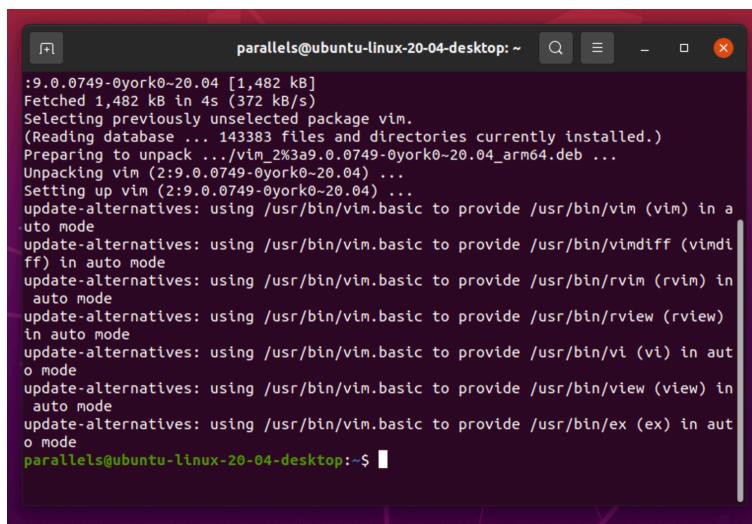
3. Perbarui informasi paket dengan perintah `sudo apt update`



```
parallels@ubuntu-linux-20-04-desktop:~$ sudo apt update
Hit:1 http://ports.ubuntu.com/ubuntu-ports focal InRelease
Hit:2 http://ports.ubuntu.com/ubuntu-ports focal-updates InRelease
Hit:3 http://ppa.launchpad.net/jonathonf/vim/ubuntu focal InRelease
Hit:4 http://ports.ubuntu.com/ubuntu-ports focal-backports InRelease
Hit:5 http://ports.ubuntu.com/ubuntu-ports focal-security InRelease
Hit:6 http://ppa.launchpad.net/neovim-ppa/stable/ubuntu focal InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
360 packages can be upgraded. Run 'apt list --upgradable' to see them.
parallels@ubuntu-linux-20-04-desktop:~$
```

Memperbarui informasi paket

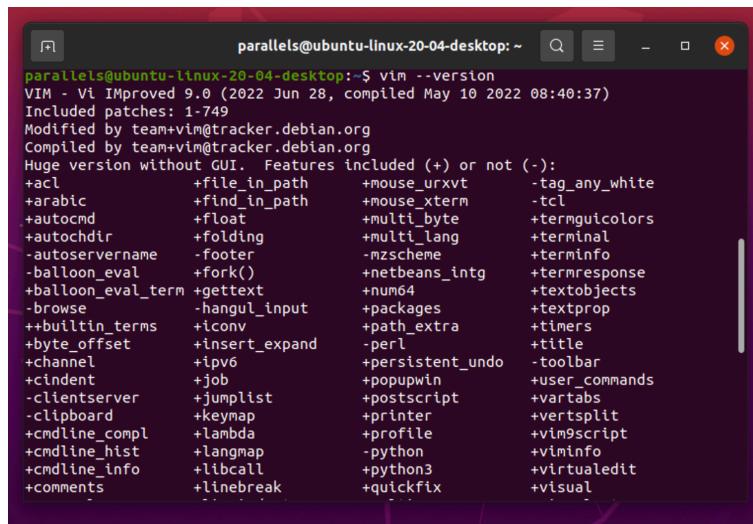
4. Sekarang kita dapat memasang Vim malalui APT dengan perintah `sudo apt install vim`



```
:9.0.0749-0york0-20.04 [1,482 kB]
Fetched 1,482 kB in 4s (372 kB/s)
Selecting previously unselected package vim.
(Reading database ... 143383 files and directories currently installed.)
Preparing to unpack .../vim_2%3a9.0.0749-0york0-20.04_arm64.deb ...
Unpacking vim (2:9.0.0749-0york0-20.04) ...
Setting up vim (2:9.0.0749-0york0-20.04) ...
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/vim (vim) in a
auto mode
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/vimdiff (vimdi
ff) in auto mode
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/rvim (rvim) in
auto mode
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/rview (rview)
in auto mode
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/vi (vi) in aut
o mode
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/view (view) in
auto mode
update-alternatives: using /usr/bin/vim.basic to provide /usr/bin/ex (ex) in aut
o mode
parallels@ubuntu-linux-20-04-desktop:~$
```

Memasang Vim

5. Verifikasi pemasangan Vim dengan perintah `vim --version` untuk mengetahui versi Vim yang terpasang



```
parallels@ubuntu-linux-20-04-desktop:~$ vim --version
VIM - Vi IMproved 9.0 (2022 Jun 28, compiled May 10 2022 08:40:37)
Included patches: 1-749
Modified by team+vim@tracker.debian.org
Compiled by team+vim@tracker.debian.org
Huge version without GUI. Features included (+) or not (-):
+acl          +file_in_path    +mouse_urxvt      -tag_any_white
+arabic       +find_in_path    +mouse_xterm      -tcl
+autocmd      +float          +multi_byte       +termguicolors
+autochdir    +folding         +multi_lang       +terminal
-autoservername -footer        -mzscheme        +terminfo
-balloon_eval  +fork()         +netbeans_intg   +termresponse
+balloon_eval_term +gettext       +num64           +textobjects
+browse       -hangul_input    +packages         +textprop
++builtin_terms  +iconv          +path_extra       +timers
+byte_offset   +insert_expand   -perl            +title
+channel      +ipv6           +persistent_undo -toolbar
+cindent       +job             +popupwin        +user_commands
-clientserver  +jumplist        +postscript       +vartabs
-clipboard     +keymap          +printer         +vertsplit
+cmdline_compl +lambda         +profile         +vim9script
+cmdline_hist  +langmap         +python          +viminfo
+cmdline_info  +libcall         +python3         +virtualedit
+comments     +linebreak       +quickfix        +visual
```

Versi Vim yang terpasang

# Mempelajari Vim

Pada bagian ini, kita akan mempelajari Vim mulai dari membuka Vim hingga menyesatkan diri ke Vim dan tidak pernah bisa keluar darinya.

Kita akan menggunakan pendekatan filosofis dengan memahami struktur bahasa Vim yang disebut dengan perintah dibanding dengan melihat daftar perintah-perintah Vim dan kemudian menghafalnya.

Menggunakan pendekatan ini dimaksudkan untuk lebih memudahkan dalam memahami Vim. Dengan memahami struktur dari suatu perintah dapat merangsang kita lebih kreatif lagi dalam mengkombinasikan satu perintah dengan perintah yang lainnya.

Sebagai contoh, jika kita hanya menghafal perintah `diw` sebagai perintah untuk menghapus sebuah "kata" di Vim, maka kita tidak pernah terpikir untuk memodifikasinya menjadi perintah untuk menghapus objek lain.

Padahal, jika dibedah, `diw` itu gabungan antara *operator* `d` untuk *delete* dan *motion* `iw` untuk *inner-word*. Jika digabung, dapat dibaca sebagai *delete inner-word*.

Berbeda bila kita memahaminya secara filosofis, kita akan terangsang untuk berpikir bahwa `iw` dapat diubah dengan *motion* yang lain sehingga dapat menghapus objek yang lain.

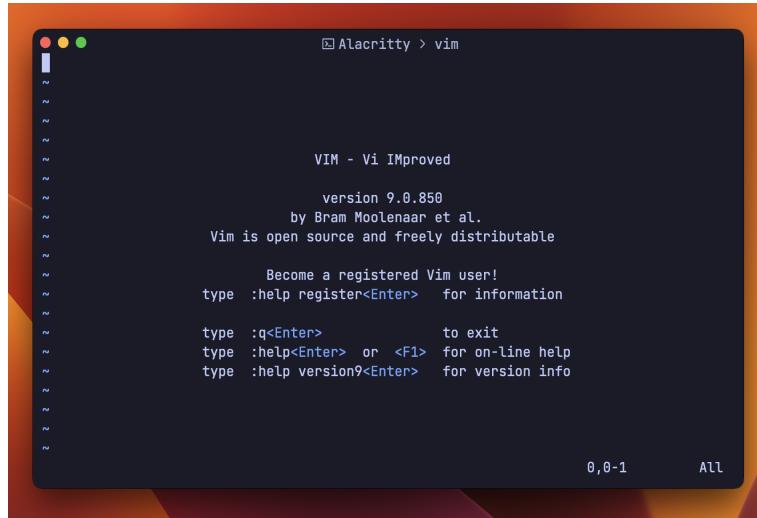
Tentu *operator* dan *motion* akan kita bahas lebih lengkap pada bagianya masing-masing. Untuk itu, kita akan mengawali pelajaran Vim dengan hal yang paling mendasar sebelum pergi pada sesuatu yang lebih spesifik.

## Membuka Dan Keluar Dari Vim

Untuk membuka Vim, kita perlu membuka aplikasi *terminal emulator* terlebih dahulu. Setelah aplikasi terminal terbuka, maka kita bisa

mengetik perintah `vim` di dalamnya dan tekan tombol `ENTER` untuk memulai menggunakan Vim.

Setelah Vim terbuka, seharusnya kamu akan melihat tampilan “*welcome screen*” dari Vim.



The screenshot shows the Vim welcome screen in a dark-themed terminal window titled "Alacritty > vim". The text displayed is:

```
VIM - Vi IMproved
version 9.0.850
by Bram Moolenaar et al.
Vim is open source and freely distributable

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

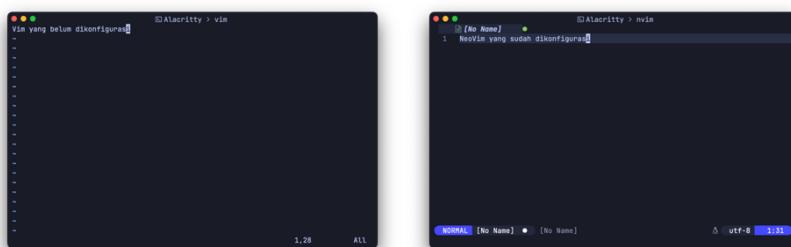
      type :q<Enter>          to exit
      type :help<Enter> or <F1> for on-line help
      type :help version9<Enter> for version info

0,0-1      All
```

Tampilan awal Vim

Pada tampilan tersebut kita bisa melihat terdapat beberapa informasi yang tersedia. Mulai dari nama program, yaitu Vim; versi Vim yang kita gunakan; sampai beberapa informasi perintah yang kita bisa gunakan.

Mungkin kamu menyadari perbedaan Vim yang saat ini kita buka dengan Vim hasil tangkapan layar saya pada awal buku ini.



Perbandingan antara Vim yang belum dikonfigurasi dan sudah dikonfigurasi

Seperti yang sudah kita bahas pada bagian awal buku prihal konfigurasi, ini bukan masalah karena saya menggunakan Neovim, ini murni karena Vim yang saat ini kita buka adalah Vim bawaan dan belum dikustomisasi sama sekali. Sedangkan pada tangkapan layar sebelum-sebelumnya itu adalah Vim yang saya sudah konfigurasi.

 Jangan sungkan untuk kembali membaca bagian awal buku ini prihal konfigurasi untuk mengingatnya kembali.

Saat ini Vim masih dalam keadaan terbuka. Kamu tidak perlu panik. Karena untuk keluar dari Vim itu sudah diberi tahu caranya di layar tersebut. Cukup gunakan perintah :q dan tekan ENTER maka kamu akan keluar dari Vim.



The screenshot shows a terminal window titled "Alacritty > vim". The terminal displays the Vim startup message, which includes the title "VIM - Vi IMproved", version "version 9.0.850", author "by Bram Moolenaar et al.", and the statement "Vim is open source and freely distributable". Below this, there is a section of text providing help information for Vim, specifically for poor children in Uganda. It includes three types of commands: "type :help iccf<Enter>" for information, "type :q<Enter>" to exit, "type :help<Enter> or <F1> for on-line help", and "type :help version9<Enter>" for version info. At the bottom of the terminal window, the cursor is positioned at the prompt ":q" followed by a small vertical bar.

Mengetik perintah :q pada Vim

Ketika mengetik tanda titik dua, kurSOR akan otomatis pindah ke bawah bagian Vim atau disebut juga dengan *command-line mode*. Apabila kamu tidak sengaja menekan perintah yang membuat kurSOR berada di *command-line*, tekan ESC atau CTRL-c untuk mengembalikan ke tempat semula.

Sejauh ini kita sudah belajar cara membuka Vim dan keluar dari Vim. Ini terlihat sederhana, tapi kamu perlu berbangga diri karena

ada sekitar 2 juta orang yang kebingungan hanya untuk keluar dari Vim<sup>10</sup>.

## Membuka Berkas Dengan Vim

Bila kamu sudah keluar dan hendak membuka Vim kembali, kamu dapat menggunakan perintah seperti sebelumnya. Namun, karena sekarang seharusnya kamu sudah memiliki satu berkas, maka kamu dapat membuka berkas tersebut secara langsung dengan Vim.

Katakanlah terdapat berkas dengan nama *welcome.js* yang berisi:

```
const userName = "Nauval";  
  
function welcome(name) {  
  
    return `Welcome, ${name}`;  
  
}  
  
console.log(welcome(userName));
```

Ketik `vim` namaberkas untuk membukanya. Karena nama berkasnya *welcome.js*, jadi kita menggunakan perintah `vim welcome.js`.

---

<sup>10</sup> "How do I exit Vim?" <https://stackoverflow.com/questions/11828270/how-do-i-exit-vim>

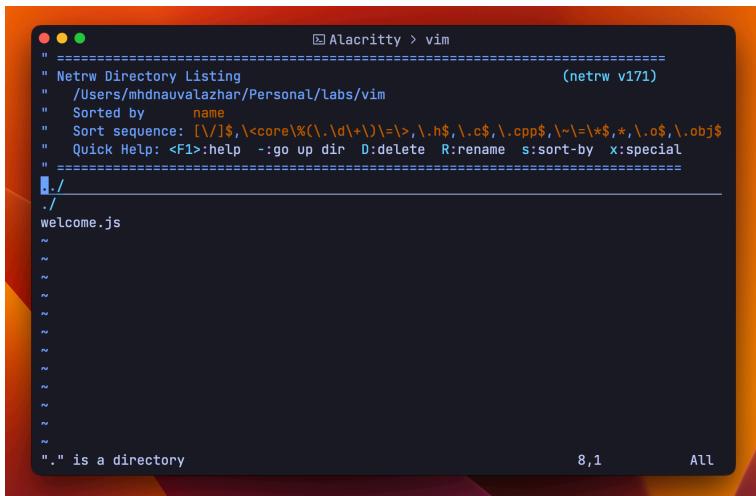


```
const userName = "Nauval";
function welcome(name) {
    return `Welcome, ${name}`;
}
console.log(welcome(userName));
~
```

Membuka berkas dengan Vim

Umumnya, Vim akan memberikan *syntax highlighter*. Bila fitur tersebut tidak aktif, kita dapat mengaktifkannya secara mandiri. Ketik :syntax on untuk mengaktifkannya.

Selain itu, kita juga dapat membuka satu folder saat ini ke dalam Vim menggunakan perintah vim ..



```
" =====
" Netrw Directory Listing                               (netrw v171)
" /Users/mhdnaualzhar/Personal/labs/vim
" Sorted by      name
" Sort sequence: [\/]$, \core\%(\.\d\+)\=\>, \h$, \c$, \cpp$, \~\=*$, *, \.o$, \.obj$ 
" Quick Help: <F1>:help  :go up dir  D:Delete  R:rename  s:sort-by  x:special
" =====
./
./welcome.js
~
```

Membuka folder dengan Vim

Ketika membuka folder dengan Vim, maka Vim akan menampilkan isi dari folder tersebut dengan *plugin* netrw yang sudah tersedia secara

bawaan di dalam Vim. Hal ini berguna ketika kita hendak menyunting beberapa berkas sekaligus di Vim.

## Mengetik Kode di Vim

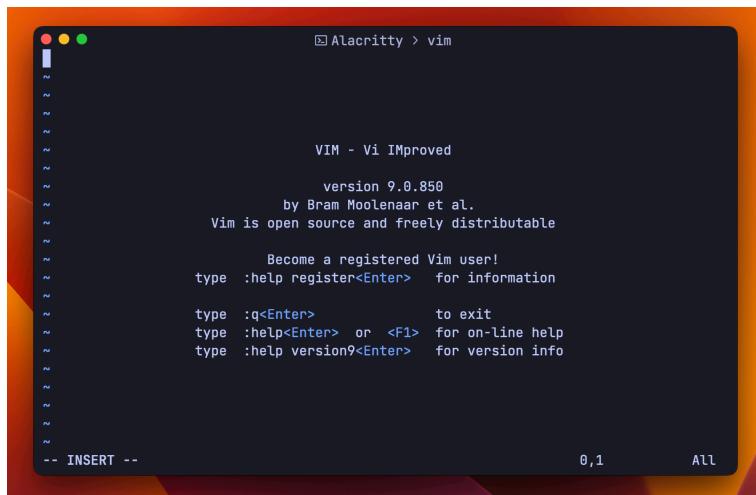
Kita buka kembali program Vim di terminal dengan cara yang sama, yaitu ketik `vim` dan tekan ENTER.



Membuka kembali Vim

Ketika Vim terbuka, maka Vim sudah pada mode normal. Pada mode normal, tombol *keyboard* yang kita tekan di dalam Vim akan diterjemahkan sebagai perintah, bukan untuk ditampilkan pada layar.

Kita bisa tekan tombol `i` pada *keyboard* untuk beralih ke mode *insert*.

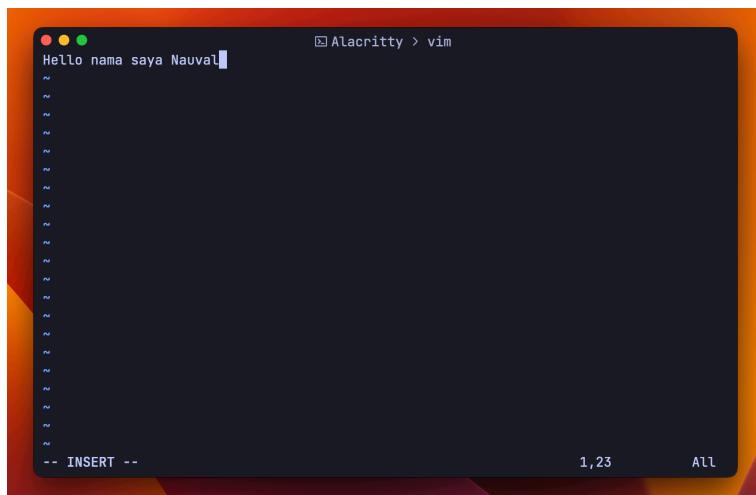


VIM - Vi IMproved  
version 9.0.850  
by Bram Moolenaar et al.  
Vim is open source and freely distributable  
Become a registered Vim user!  
type :help register<Enter> for information  
type :q<Enter> to exit  
type :help<Enter> or <F1> for on-line help  
type :help version9<Enter> for version info

Beralih ke mode *insert*

Ketika berada di mode *insert* maka terdapat indikator "--INSERT--" yang bisa kita lihat di pojok-kiri-bawah tampilan Vim. Ini untuk memberitahu kita bahwa Vim masih berada di mode *insert*. Pada mode *insert* kita bisa mengetik karakter yang kita mau seperti pada editor teks *non-modal*.

Dalam mode ini, kita sudah dapat mengetik apa saja seperti pada editor teks konvensional.



Hello nama saya Nauval

Mengetik di dalam Vim

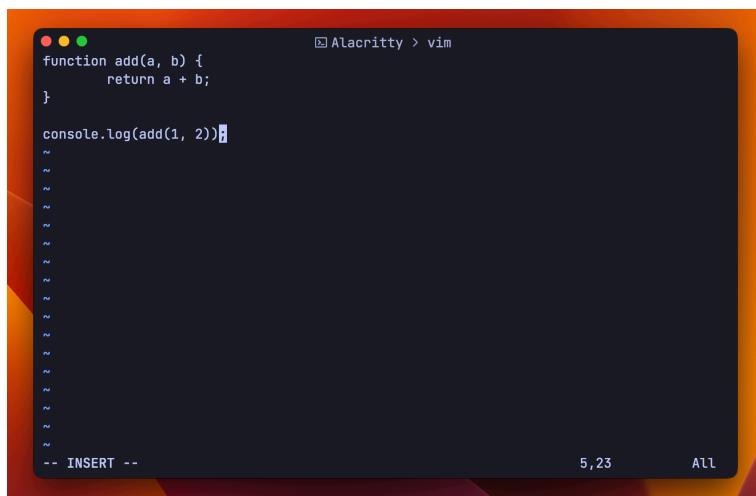
Selain menggunakan perintah `i`, kamu dapat menggunakan beberapa perintah berikut untuk beralih ke mode *insert* dan sekaligus memindahkan posisi kursor:

- `I` memindahkan kursor ke awal baris
- `A` memindahkan kursor ke akhir baris
- `a` memindahkan kursor satu karakter ke depan
- `o` memindahkan kursor ke bawah (menambah baris baru)
- `O` memindahkan kursor ke atas (menambah baris baru)

Kita dapat kembali ke mode normal dengan menggunakan tombol `ESCAPE`.

Ketika kita berada di mode normal, maka indikator yang sebelumnya terlihat akan hilang. Ini artinya kita sudah berada di mode normal kembali. Tombol *keyboard* yang ditekan akan diterjemahkan menjadi perintah oleh Vim.

Sekarang alihkan lagi Vim ke mode *insert* dan kita akan coba menulis kode. Kamu boleh menulis kode apapun. Sebagai contoh, saya akan menulis kode JavaScript singkat.

A screenshot of a terminal window titled "Alacritty > vim". The terminal is displaying a block of JavaScript code:

```
function add(a, b) {  
    return a + b;  
}  
  
console.log(add(1, 2));
```

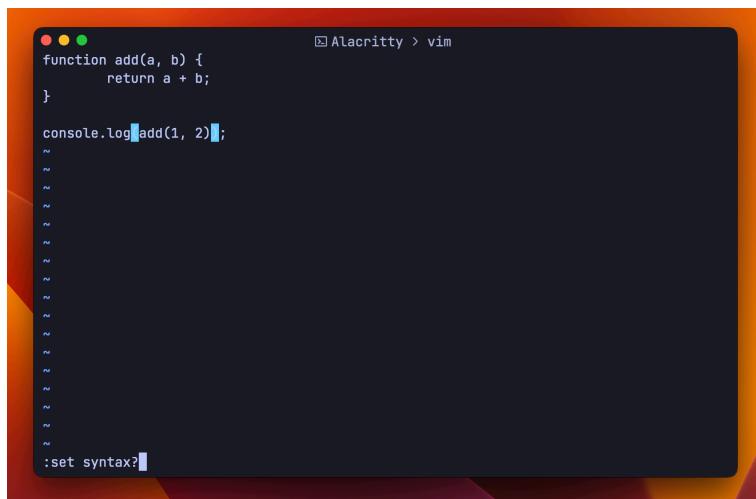
The status bar at the bottom shows "- INSERT -" on the left, "5,23" in the center, and "All" on the right. The background of the terminal is dark, and the text is white.

Menulis kode JavaScript di Vim

Sampai sini mungkin kamu menyadari ada yang kurang? Ya, tidak ada *syntax highlighting*.

## Mengatur Syntax Highlighter

Kita dapat memeriksa terlebih dahulu status dari fitur *syntax highlighter* di Vim, untuk memeriksanya gunakan perintah `:set syntax?`. Sebelum mengetik perintah tersebut, pastikan Vim sedang berada di mode normal.

A screenshot of a terminal window titled "Alacritty > vim". The terminal displays the following code:

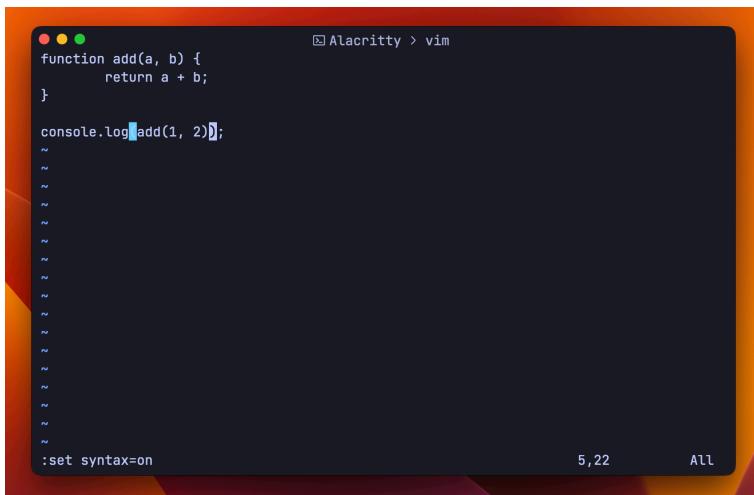
```
function add(a, b) {
    return a + b;
}

console.log(add(1, 2));
```

The word "add" is highlighted in blue, indicating that the syntax highlighter is active. Below the code, the command `:set syntax?` is entered in the terminal.

Memeriksa fitur syntax highlighter

Bila hasil yang keluar selain `syntax=on`, maka dapat disimpulkan fitur tersebut sedang tidak aktif. Untuk mengaktifkannya, gunakan perintah `:set syntax=on`.



A screenshot of a terminal window titled "Alacritty > vim". The terminal displays the following code:

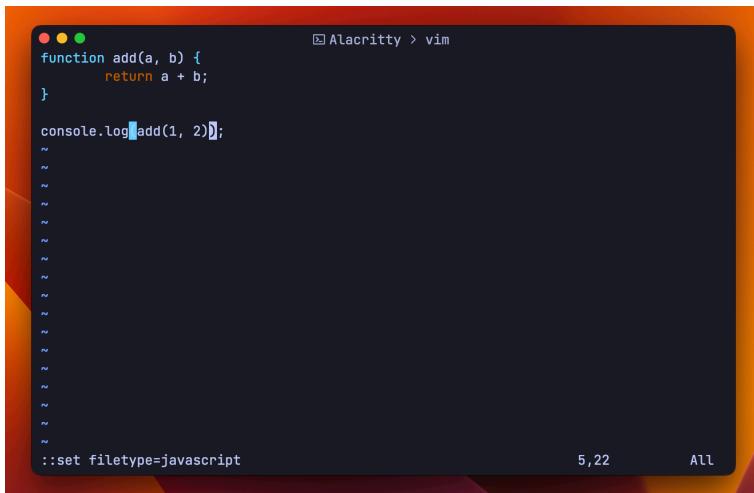
```
function add(a, b) {
    return a + b;
}

console.log(add(1, 2));
```

The code uses color-coded syntax highlighting: "function", "add", "a", "b", "return", "console", "log", and the numbers "1" and "2" are highlighted in various colors. The terminal status bar at the bottom shows the command ":set syntax=on", the current line number "5,22", and the search term "All".

Mengaktifkan fitur *syntax highlighter*

Kita baru saja mengaktifkan fitur *syntax highlighter* di Vim, namun belum memberitahu jenis kode yang kita tulis. Untuk memberitahu Vim jenis kode yang kita tulis adalah JavaScript, gunakan perintah `:set filetype=javascript`.



A screenshot of a terminal window titled "Alacritty > vim". The terminal displays the same JavaScript code as the previous screenshot. The syntax highlighting is identical. The terminal status bar at the bottom shows the command `::set filetype=javascript`, the current line number "5,22", and the search term "All".

Mengubah jenis berkas

Mungkin sampai sini kamu akan merasa bahwa Vim semakin *ribet*. Karena untuk "mewarnai" sintaksis saja perlu dilakukan secara

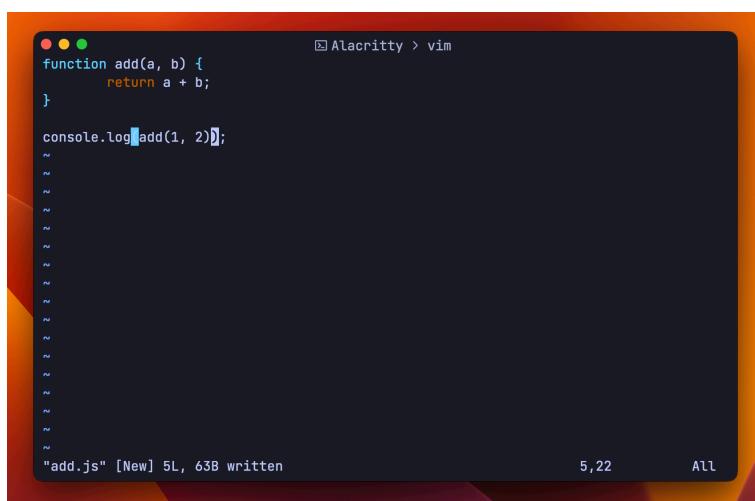
manual. Umumnya, hal serupa bisa kamu temui di editor teks konvensional, karena pada dasarnya editor teks menganggap karakter yang kamu tulis hanyalah *plain text*, bukan bahasa pemrograman spesifik.

Sebetulnya, Vim akan mengenali jenis sintaksis dari kode yang kita tulis berdasarkan nama berkasnya. Seandainya kode yang kita tulis sebelumnya kita simpan terlebih dahulu, maka Vim dapat –secara otomatis– mengetahui jenis sintaksis pada kode tersebut tanpa harus diberitahu secara eksplisit.

Selain itu ketika menyimpan kode yang kita tulis ke dalam sebuah berkas, Vim juga akan secara otomatis mengubah jenis sintaksis secara otomatis sesuai nama berkasnya. Pada kasus nyata, kita tidak akan mengubah jenis sintaksis secara manual kecuali pada kasus-kasus tertentu, seperti *debugging* misalnya.

## Menyimpan Berkas

Kode yang kita tulis sebelumnya itu belum disimpan ke dalam berkas apapun, itu kenapa Vim tidak dapat mengenalinya. Kita dapat menyimpan kode tersebut dengan perintah `:w nameberkas`. Saya akan menggunakan `add.js` sebagai nama berkas untuk kode sebelumnya. Perintahnya akan menjadi `:w add.js`.

A screenshot of a terminal window titled "Alacritty > vim". The terminal displays the following code:

```
function add(a, b) {
    return a + b;
}

console.log(add(1, 2));
```

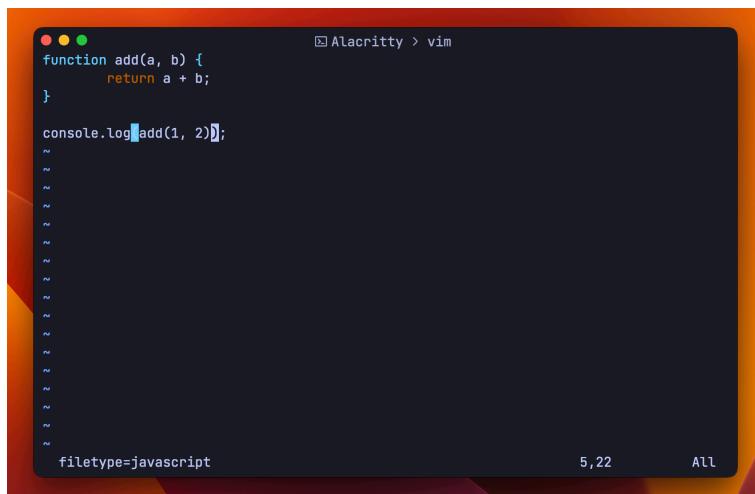
The cursor is positioned at the end of the second line. At the bottom of the terminal, the status bar shows the file name "add.js" [New], 5L, 63B written, the line number 5, 22, and the word All.

## Menyimpan berkas di Vim

Ketika berkas berhasil disimpan akan terlihat indikator di kiri-bawah bertuliskan seperti ini:

```
"add.js" [New] 5L, 63B written
```

Setelah itu kita dapat memeriksa jenis sintaksis yang sedang digunakan saat ini dengan perintah `:set filetype?.`

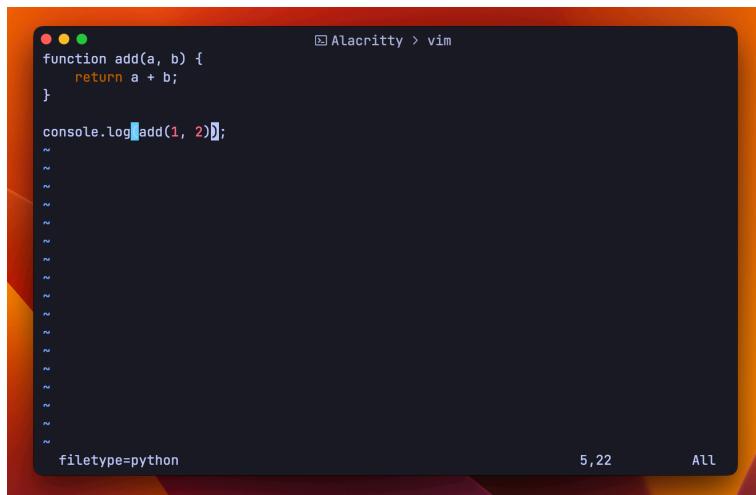
A screenshot of a terminal window titled "Alacritty > vim". The terminal displays a block of JavaScript code:

```
function add(a, b) {  
    return a + b;  
}  
  
console.log(add(1, 2));
```

The status bar at the bottom shows "filetype=javascript" on the left, "5,22" in the center, and "All" on the right.

Hasil dari perintah `:set filetype?`

Kamu dapat mengcoh Vim dengan memberikan ekstensi yang tidak sesuai dengan kode di dalam berkas tersebut. Misal, kamu menulis kode JavaScript dan berikan ekstensi `.py` pada nama berkasnya. Maka Vim akan menganggap kode JavaScript kamu sebagai kode Python.

A screenshot of a terminal window titled "Alacritty > vim". The terminal displays a block of code in Python syntax. The code defines a function named "add" that takes two arguments, "a" and "b", and returns their sum. It then calls this function with arguments 1 and 2, and prints the result. The terminal uses a dark theme with color-coded syntax highlighting. The status bar at the bottom shows "filetype=python", the current line number "5,22", and the search term "All".

```
function add(a, b) {  
    return a + b;  
}  
  
console.log(add(1, 2));
```

Kode JavaScript dengan jenis sintaksis Python

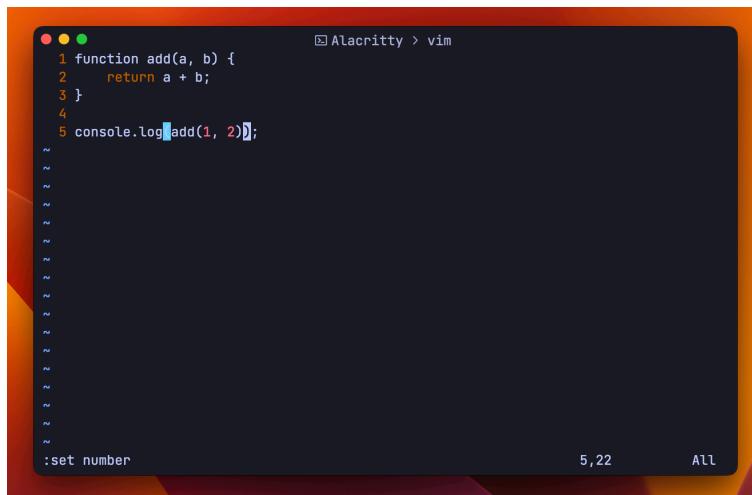
Bila kamu menemui kasus asing yang di mana Vim salah menganggap kode yang kamu tulis, kamu dapat memberitahu Vim secara eksplisit jenis sintaksis apa yang kamu inginkan seperti yang telah dijelaskan di awal.

## Menampilkan Nomor Baris

Sangat sulit bila kita menulis kode tanpa dibantu dengan adanya nomor baris. Kita tidak tahu sudah berapa baris kode yang kita tulis dan nomor baris juga berguna sebagai referensi agar kita dapat melompat ke baris kode tertentu.

Sejauh ini kita hanya memberikan warna sintaksis pada kode yang kita tulis, dan belum ada nomor baris sama sekali.

Untuk menampilkan nomor baris di Vim kita dapat menggunakan perintah `:set number`.



A screenshot of a terminal window titled "Alacritty > vim". The terminal displays the following code:

```
1 function add(a, b) {  
2     return a + b;  
3 }  
4  
5 console.log(add(1, 2));
```

The code is preceded by several blank lines, each marked with a tilde (~). At the bottom left of the terminal, the command ":set number" is visible. In the bottom right corner, the status bar shows "5,22" and "All".

Menampilkan nomor baris

Atau juga bisa menggunakan perintah :set nu untuk perintah yang lebih ringkas.

Apabila kamu memiliki alasan tertentu untuk menghilangkan nomor baris, kamu dapat menggunakan perintah:

- :set nonu
- :set nonumber
- :set nu!
- :set number!

Semua perintah tersebut akan mencapai tujuan yang sama.

Mungkin kamu akan berpikir bahwa untuk menggunakan fitur dasar editor teks saja harus menghidupkannya secara manual di Vim. Maka akan sangat merepotkan ketika hendak menggunakan Vim harus diawali dengan mengaktifkan fitur-fitur dasar ini.

Pada dasarnya, pemikiran seperti itu adalah benar. Namun, umumnya pengguna Vim menuliskan beberapa konfigurasi untuk menyalakan beberapa fitur yang diinginkan, termasuk *syntax highlighter*, dan *line numbers*. Setiap kali Vim dibuka, maka fitur-fitur tersebut tidak perlu diaktifkan secara manual lagi.

Tidak perlu heran bila kamu sudah mengatur beberapa konfigurasi seperti mengaktifkan nomor baris, fitur *syntax highlighter*, atau fitur-fitur lainnya yang akan ditemui di bagian-bagian berikutnya akan hilang atau kembali ke pengaturan bawaan ketika *program Vim* ditutup. Karena konfigurasi yang dilakukan dengan perintah-perintah tersebut tidak persisten.

Kita akan membahas perihal konfigurasi pada bagian lain, untuk saat ini kita hanya perlu memahami cara mengoperasikan Vim terlebih dahulu. Supaya paham dasar-dasar Vim dan paham cara berkomunikasi dengannya.

## Mengatur Colorscheme

Pada editor teks semacam VS Code, kita dapat mengganti-ganti tema sesuai dengan suasana hati atau hanya sesuai selera saja. Tapi, intinya kita bisa mengganti tema. Di Vim juga kita dapat melakukan hal yang sama. Vim menyebutnya *colorscheme* ketimbang *theme*.

Kita dapat menggunakan perintah `:colorscheme` di dalam Vim untuk mengetahui *colorscheme* yang sedang digunakan saat ini. Vim yang saya gunakan saat ini menggunakan *colorscheme default*.

A screenshot of a terminal window titled "Alacritty > vim". The terminal displays a snippet of JavaScript code:

```
function add(a, b) {  
    return a + b;  
}  
  
console.log(add(1, 2));
```

The status bar at the bottom shows the text "default" on the left, "5,22" in the center, and "All" on the right. The background of the terminal is dark, and the text is white and yellow.

Nama *colorscheme* yang sedang digunakan

Sebelum mengganti *colorscheme* saat ini dengan yang lain, kita perlu tahu terlebih dahulu daftar *colorscheme* yang tersedia. Setidaknya kita memiliki dua cara untuk mengetahuinya:

1. Menggunakan perintah `:colorscheme`, lalu tekan SPASI dan tekan tombol TAB
2. Menggunakan perintah `:colorscheme`, lalu tekan SPASI, lalu tekan tombol kombinasi CTRL+d



A screenshot of a terminal window titled "Alacritty > vim". The terminal displays a list of available colorschemes. At the top, there is some code: "5 console.log(add(1, 2));". Below it, there are several tilde (~) characters. Then, the command ":colorscheme" is entered, followed by a list of names: blue, delek, evening, koehler, murphy, quiet, slate, darkblue, desert, habamax, lunaperche, pablo, ron, torte, default, elflord, industry, morning, peachpuff, shine, zellner. Finally, the command ":colorscheme" is repeated again.

Daftar *colorscheme* yang tersedia

Bila kamu sudah menemukan nama *colorscheme* yang hendak digunakan, jalankan perintah `:colorscheme namacolorscheme` untuk mengubah *color scheme* di Vim saat ini. Untuk sekarang, saya akan tetap menggunakan *colorscheme default*.

## Motion

Ketika menggunakan editor teks konvensional seperti VS Code, aktivitas memindahkan kursor dengan menggunakan *mouse* adalah hal yang lazim. Berbeda dengan Vim, kamu akan melakukan aktivitas semacam itu hanya dengan *keyboard* saja. Karena memindahkan tangan dari atas *keyboard* ke atas *mouse* itu memakan waktu lama.

Perlu saya katakan sekali lagi, esensi menggunakan Vim adalah efisiensi *keyboard*. Tangan kamu tidak akan pernah pindah dari atas *keyboard* ke atas *mouse*. Kamu akan melakukan semuanya dengan *keyboard*, termasuk memindahkan posisi kursor.

Pada dasarnya, kamu dapat memindahkan posisi kursor dengan tombol panah atas, bawah, kiri, dan kanan seperti pada umumnya. Namun, umumnya pengguna Vim menggunakan tombol `hjkl` untuk memindahkan posisi kursor ke atas, ke bawah, ke atas, dan ke kanan.

- `h` untuk memindahkan kursor ke kiri
- `j` untuk memindahkan kursor ke bawah
- `k` untuk memindahkan kursor ke atas
- `l` untuk memindahkan kursor ke kanan



*Experienced users prefer the `hjkl` keys because they are always right under their fingers. Beginners often prefer the arrow keys, because they do not know what the `hjkl` keys do. The mnemonic value of `hjkl` is clear from looking at the keyboard. Think of `j` as an arrow pointing downwards.*

– Bram Moolenaar

Sebagai langkah awal, tidak masalah memindahkan kursor dengan tombol arah panah. Karena itu yang saya lakukan juga ketika awal belajar Vim. Tapi, kamu perlu membiasakan menggunakan tombol `hjkl` untuk memindahkan kursor. Semakin lama dan semakin terbiasa, tanganmu tidak akan pernah lepas di tombol tersebut. Serius!

Perlu diingat, kamu perlu berada di mode normal untuk menggunakan perintah `hjkl` untuk memindahkan kursor.

Perintah tersebut, di dalam Vim disebut juga dengan *cursor motions* atau *motion* saja. *Motion* di dalam Vim merupakan sebuah perintah untuk memindahkan posisi kursor. Perintah `hjkl` ini termasuk ke dalam *left-right* dan *up-down motion*.

Mungkin kamu bertanya-tanya mengapa Vim menggunakan hjkl. Selain karena tombol tersebut berada pada *home row keyboard*, juga karena *layout keyboard* jaman dahulu yang di mana tombol-tombol tersebut berfungsi sebagai tombol arah panah.



Sumber: <https://catonmat.net/why-vim-uses-hjkl-as-arrow-keys>

Selain *motion* hjkl, kamu juga dapat menggunakan angka 0 untuk pergi ke awal baris; tanda \$ untuk pergi ke akhir baris.

Vim memiliki jenis *motion* yang lain dan yang umumnya sering –setidaknya oleh saya– gunakan adalah:

- *word motion*
- *text object motion*
- *text object selection*
- *jumps motion*

Pada kasus yang lebih konkret, kamu jarang sekali memindahkan kursor per karakter, melainkan per kata, per baris, atau sepanjang jangkauan yang kamu tentukan seperti mulai dari tanda { hingga tanda }.

## Word Motion

Kamu dapat menggunakan *word motion* untuk memindahkan kursor per kata. Sebagai contoh, bila kamu memiliki baris kode seperti di bawah dan posisi kursor kamu saat ini berada di karakter f:

```
function welcome() {  
    //  
}
```

Kamu dapat menggunakan perintah `w` untuk memindahkan kursor satu “kata” ke depan. Maka posisi kursor kamu sekarang berada di huruf `w`:

```
function Welcome() {  
    //  
}
```

Sebuah *word* atau kata di Vim terdiri dari urutan huruf, angka dan garis bawah, atau urutan karakter *non-blank* lainnya, dipisahkan dengan spasi (spasi, tab, *end-of-line*).

Kebalikannya, kamu dapat menggunakan `b` untuk pidah satu “kata” ke belakang. Selain menggunakan `w` dan `b`, kamu juga dapat menggunakan `e` dan `ge`. Bedanya, kamu bisa lihat kode sebelumnya:

```
function welcome() {  
    //  
}
```

Bila kursor kamu saat ini berada di karakter `f`, bila kamu menggunakan perintah `e`, maka kursor akan pindah ke karakter `n`.

```
function Welcome() {  
    //  
}
```

Karena perintah `e` untuk memindahkan kursor ke akhir kata.

## Text Object Motion

Vim juga memungkinkan kamu memindahkan posisi kursor per kalimat dan juga per paragraf. Kamu membutuhkan *text object motion* untuk melakukannya. Sebagai contoh, kamu memiliki teks seperti ini:

**L**orem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book.

It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Bila kursor kamu saat ini berada di huruf *L* pada kata "Lorem", kamu dapat menggunakan tanda kurung ) untuk melompat satu kalimat ke depan. Maka saat ini kursor kamu akan berada di huruf *L* pada kata "Lorem" di kalimat kedua.

**L**orem Ipsum is simply dummy text of the printing and typesetting industry. **L**orem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book.

It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Kamu dapat menggunakan tanda kurung ( untuk kembali satu kalimat ke belakang. Maka posisi kursor saat ini berada di posisi awal kalimat.

Sebuah kalimat didefinisikan sebagai diakhiri dengan titik, tanda seru atau tanda tanya, diikuti oleh *end-of-line*, spasi, atau tab.

Selain itu, kamu juga dapat menggunakan tanda kurung kurawal { dan } untuk memindahkan kursor per paragraf. Kita bisa lihat lagi teks yang kita miliki seperti ini:

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book.

It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Apabila menggunakan tanda kurung kurawal }, maka posisi kursor saat ini berada di bawah paragraf pertama.

Iorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book.

I

It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

Paragraf bukan hanya pada teks biasa saja, “paragraf” di Vim diawali setelah setiap baris kosong. Kode definisi fungsi di bawah dianggap “paragraf” oleh Vim:

```
function foo() {  
}
```

Kamu dapat menggunakan tanda kurung kurawal { untuk kembali satu paragraf ke belakang.

## Text Object Selection

Berbeda dengan *motion* sebelumnya, *text object selection* hanya dapat digunakan pada mode visual atau setelah perintah *operator*. Di awal tulisan ini saya memberi contoh kasus mengubah teks di dalam tag HTML dengan menggunakan perintah `cit` di *keyboard*.

Perintah `cit` tersebut sebetulnya merupakan sebuah kombinasi antara *operator* dan *text object selection*.

- `c` adalah *operator*
- `it` adalah *text object selection*

Perintah `c` adalah *operator* yang memberitahu Vim untuk menghapus teks dan masuk ke dalam mode *insert*. Sedangkan perintah `it` adalah *text object selection* yang memberitahu jangkauan teks yang harus dihapus, dalam konteks ini jangkaunnya adalah teks yang ada di dalam *tag*.

Itu kenapa ketika kita memasukkan perintah `cit` akan menghapus teks di dalam *tag* dan langsung masuk ke dalam mode *insert*. Kita akan membahas *operator* lebih jauh di bagian berikutnya.

Seperti yang saya katakan di awal, *text object selection* hanya dapat digunakan pada mode visual dan setelah perintah *operator*. Kamu tidak dapat menggunakan *text object selection* pada mode normal.

Selain itu, terdapat beberapa *text object selection* lainnya yang dapat digunakan: `iw`, `aw`, `at`, `ab`, `ip`, dan yang lainnya. Untuk saat ini mungkin kamu tidak perlu memahami semua itu, kamu hanya perlu tahu eksistensi dan kegunaan *text object selection* di Vim.

## Jump Motion

Semakin besar aplikasi yang kita buat, semakin besar juga basis kodennya. Katakanlah kamu memiliki kode JavaScript yang di dalamnya terdapat sebuah fungsi dan beberapa baris *statement* sehingga *keyword return* terlalu jauh untuk dijangkau.

```
function findMyLover() {  
    // kode  
    // yang  
    // sangat  
    // panjang  
  
    return null;  
}
```

Seandainya kursor kamu saat ini berada di tanda kurung kurawal `{` dan ingin mengganti nilai `return`, tentu kamu dapat menggunakan tombol `j` untuk terus ke bawah hingga mencapai posisi yang diinginkan. Tapi itu bukan cara menggunakan Vim yang efisien.

```
function findMyLover() {  
    // kode  
    // yang  
    // sangat  
    // panjang  
  
    return null;  
}
```

Kamu dapat menggunakan tombol `%` untuk langsung melompat ke tanda kurung kurawal `}%`.

```
function findMyLover() {  
    // kode  
    // yang  
    // sangat  
    // panjang  
  
    return null;  
}
```

Ini tidak terbatas pada tanda kurung kurawal saja, kamu dapat melakukan hal yang sama pada tanda kurung (), tanda kurung siku [], juga C-style comment /\* ... \*/.

Sebelumnya kita sudah membahas cara memindahkan kursor per paragraf dan per kalimat. Faktanya, perintah tersebut juga termasuk ke dalam *jump motion* pada Vim. Perintah yang termasuk ke dalam *jump* adalah “““, “~”, “G”, “/”, “?”, “n”, “N”, “%”,“(, “)”, “[, “]”, “{, “}”, “:s”, “:tag”, “L”, “M”, “H”.

Pada bagian *motion* ini kita sudah belajar bagaimana memindahkan kursor di Vim. Pada dasarnya kita dapat menggunakan tombol arah panah, namun kita juga dapat menggunakan beberapa *motion* di atas untuk mempercepatnya. Hal ini tentu akan membuat repot bila belum terbiasa, maka biasakanlah!

Kamu tidak perlu memaksakan menggunakan semua *motion* di atas, bila belum terbiasa tidak apa-apa menggunakan tombol arah panah untuk memindahkan kursor. Kamu dapat mencobanya secara bertahap, itu yang saya lakukan juga ketika saya belajar Vim. Tidak usah minder!

## Operator

Pada dasarnya, menghapus teks pada Vim sama seperti editor teks pada umumnya. Kita dapat menggunakan tombol BACKSPACE atau DELETE di macOS. Namun, bila kita mencoba menekan BACKSPACE di

mode normal, maka teks tidak akan terhapus. Hal ini hanya berlaku di mode *insert* saja.

Apakah harus pergi ke mode *insert* terlebih dahulu untuk menghapus teks di Vim? Jawabannya adalah iya bila kamu ingin merepotkan diri sendiri. Vim memiliki perintah untuk menghapus teks pada mode normal.

Sebenarnya, ketika kamu di Vim kamu akan selalu berada di mode normal. Kamu akan berpindah ke mode lain sesekali saja ketika memang membutuhkannya. Sebagai contoh, kamu akan pergi ke mode *insert* untuk menulis teks atau kode. Setelah menuliskannya, kamu akan kembali ke mode normal. Hal seperti ini akan menjadi *habit*.

Di Vim perintah untuk menghapus adalah `d`. Ketika kamu menekan tombol `d` di *keyboard* maka tidak akan terjadi apa-apa. Karena perintah `d` tersebut adalah *operator*. Di Vim, *operator* menunggu perintah lain untuk melaksanakan tugasnya.

Dalam kata lain, ketika kamu menekan tombol `d`, kamu hanya memberitahu Vim bahwa kamu *hendak* menghapus sesuatu, tapi kamu tidak memberitahu Vim mengenai apa yang hendak dihapus. Vim pasti bingung, *'kan?* Pada saat itulah *motion* berguna. Kamu dapat menggunakan salah satu *motion* yang kamu perlukan untuk menghapus teks.

Sebagai contoh, kamu dapat menggunakan `w` untuk menghapus satu “kata” ke depan dari posisi kursor saat ini. Bila dikombinasikan, maka perintahnya menjadi `dw`. Perintah `dw` berarti *delete word*.



Sebagai informasi, kamu dapat menggunakan perintah `x` untuk menghapus satu karakter di bawah posisi kursor saat itu. Juga dapat menggunakan `s` untuk sekaligus masuk ke mode *insert*.

Bukan hanya `w`, kita belajar *motion* yang lain juga. Kita dapat menggunakan *motion* yang kita telah pelajari sebelumnya sebagai kombinasi dengan *operator delete*.

Untuk menghapus satu “kata” ke belakang, misalnya, kita dapat menggunakan `db`. Begitu juga dengan *motion* yang lain, seperti:

- `d}` menghapus satu paragraf ke depan
- `d{` menghapus satu paragraf ke belakang
- `dit` menghapus teks di dalam *tag block*
- `d)` menghapus satu kalimat ke depan
- `d(` menghapus satu kalimat ke belakang
- `d%` menghapus teks dalam cakupan tanda yang ditemukan (contoh: `()`)
- dan seterusnya

Ya, itulah bahasa Vim. Bahasanya lebih ringkas dari bahasa manusia, namun bagian sulitnya adalah mencoba mengingatnya. Bagaimana tidak ringkas, tata bahasa Vim hanya terdiri dari: kata kerja + kata benda.

Kata kerja di Vim itu diisi dengan *operator*:

- `d` untuk menghapus
- `c` untuk menghapus dan masuk ke mode *insert*
- `y` untuk yank (menyalin/*copy*)
- `>` untuk indentasi

Sedangkan kata benda di Vim itu diisi dengan *motion*:

- `w` untuk satu kata ke depan
- `b` untuk satu kata ke belakang
- `it` untuk dalam *tag block*
- dan sebagainya

Sampai sini mungkin intelegensi kamu dapat memahami dari contoh sebelumnya bahwa perintah `d` dapat diganti dengan `c`, `y` atau *operator* lainnya. Ya, memang benar! Karena begitulah cara kita berkomunikasi dengan Vim.

Mungkin kamu akan menggunakan `cw` untuk menghapus “kata” dan beralih ke mode *insert*; mungkin kamu menggunakan `y`) untuk menyalin satu “kalimat” ke depan; mungkin kamu menggunakan `>}` untuk menambah indentasi satu “paragraf” ke depan. Begitu seterusnya.

Bila kamu mengulang *operator* sebagai kata benda di Vim, maka Vim akan melakukan sesuatu pada satu baris di mana kurSOR berada. Misal, seperti ini:

- `dd` akan menghapus satu baris
- `yy` akan menyalin satu baris
- `cc` akan menghapus satu baris dan masuk ke mode *insert*
- `>>` akan memberikan indentasi pada satu baris

Selain itu, kamu juga dapat mengulang *operator* atau *motion* dengan menambahkan angka di awal. Misal, kamu dapat menggunakan `3dd` untuk mengulang perintah `dd` sebanyak 3 kali. Begitu juga dengan *motion*, kamu dapat menggunakan `8j` untuk memindahkan kurSOR 8 kali ke bawah.

Tidak hanya itu, seandainya kamu ingin menghapus 2 kata ke depan, kamu dapat menggunakan perintah `d2w` yang berarti *delete 2 words*. Atau kamu juga dapat menggunakan `4d2w` yang berarti 4 kali *delete 2 words*.

Kamu hanya perlu mengkombinasikan *operator* dengan *motion* agar Vim dapat melakukan sesuatu yang kamu inginkan.

Hal yang perlu kamu ketahui adalah ketika kamu menggunakan *operator* untuk menghapus teks, seperti `d` atau `c`, *operator* juga akan menaruh teks yang dihapus ke dalam ruang memori di Vim benama *registers*.

Jadi saat kamu menghapus teks dengan *operator* `d`, misalnya. Teks tersebut akan hilang, dan teks tersebut juga akan ditaruh ke dalam *registers*, jadi saat kamu melakukan paste, maka akan muncul teks terakhir yang kamu hapus. Kita akan belajar mengenai *registers* di bagian berikutnya.

Sampai sejauh ini kita sudah belajar hal yang paling penting di Vim, yaitu *operator* dan *motion*. Karena dengan keduanya kita dapat berkomunikasi dengan Vim.

## Copy dan Paste

Vim menggunakan istilah *yank* untuk menyalin, itu kenapa perintahnya adalah `y`. Sama seperti *operator* yang lainnya, kamu juga dapat mengkombinasikan *motion* dengan *yank*. Misal, untuk menyalin “kata” kamu dapat menggunakan `yw`; untuk menyalin paragraf kamu dapat menggunakan `y}`; dan sebagainya.

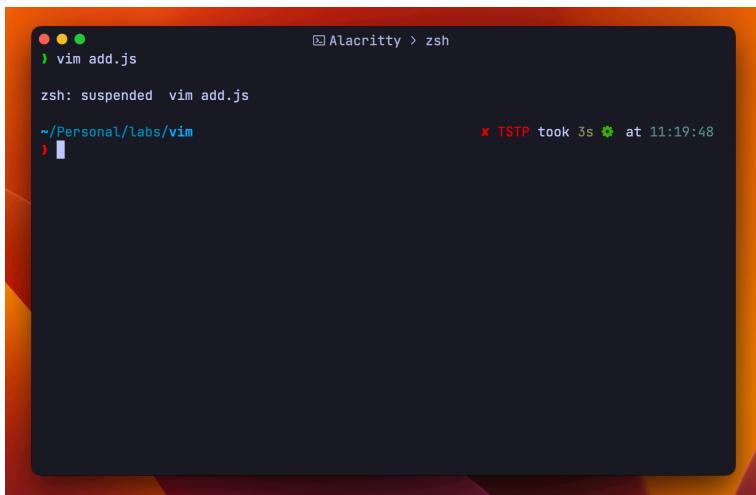
Sedangkan untuk menempelkan hasil yang disalin, kamu dapat menggunakan perintah `p`.

## Undo, Redo dan Repeat

Kita tidak mungkin tidak membuat kesalahan dalam menulis kode. Sama seperti editor teks lainnya, Vim juga pemaaf dan memberikanmu kesempatan untuk memperbaikinya.

Untuk meng-*undo* di Vim, kamu dapat menggunakan `u`. Kontras, untuk me-*redo* di Vim, kamu dapat menggunakan `ctrl+r`.

Bila kamu menjalankan Vim di *terminal emulator*, dan kamu mencoba menekan `ctrl+z` dengan harapan untuk meng-*undo*, maka yang terjadi adalah *shell* akan men-suspend program Vim yang sedang berjalan dan menaruhnya di *background process*.

A screenshot of a terminal window titled "Alacritty > zsh". The window has a dark background with orange and red highlights. The text inside shows:

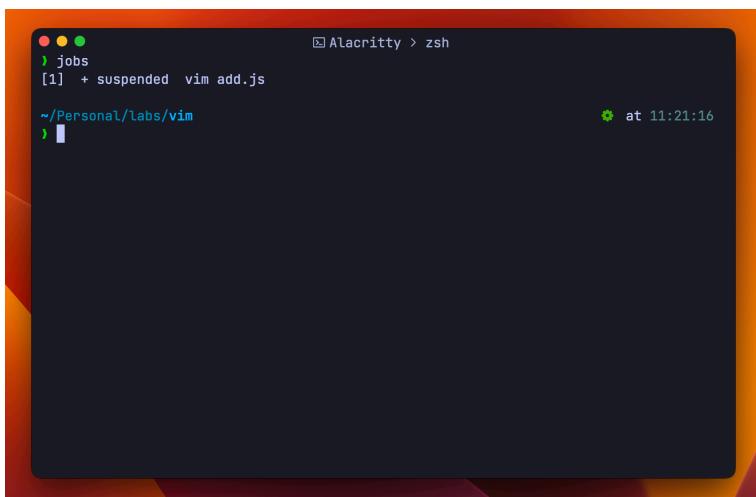
```
vim add.js
zsh: suspended vim add.js
~/Personal/labs/vim
x TSTP took 3s at 11:19:48
```

The status bar at the bottom right shows "at 11:19:48".

Vim ter-suspend

Kamu dapat menggunakan perintah `fg` untuk mengembalikannya. Perintah tersebut akan membawa *background process* terakhir ke *foreground* sehingga kamu dapat menggunakan Vim kembali.

Apabila tidak berhasil, kamu dapat menggunakan perintah `jobs` untuk mengetahui *program* yang ada di *background*.

A screenshot of a terminal window titled "Alacritty > zsh". The window has a dark background with orange and red highlights. The text inside shows:

```
jobs
[1] + suspended vim add.js
~/Personal/labs/vim
* at 11:21:16
```

The status bar at the bottom right shows "at 11:21:16".

Memeriksa program yang berjalan di background

Seandainya kamu memiliki beberapa daftar dan kamu menemukan *program* Vim yang tidak sengaja kamu *suspend*. Kamu dapat

menggunakan nomor di dalam tanda kurung siku [ ] sebagai referensi perintah fg. Sebagai contoh, jadinya seperti ini: fg %1.

Balik lagi ke Vim.

Vim memiliki perintah yang --menurut saya-- canggih, yaitu perintah *dot* atau titik: .. Perintah . akan mengulang perintah terakhir yang kamu gunakan. Apabila perintah terakhir yang kamu gunakan adalah dd maka perintah . akan melakukan peran yang sama.

Perintah ini sungguh *powerful!* Sebagai contoh, saya ingin mengganti nama fungsi dari add menjadi tambah. Saya dapat menggunakan ciw pada kata “add” sekali, dan tekan . di kata “add” yang lain.

Itu hanya salah satu contoh dari sekian banyak contoh *repeatable command* yang kamu akan temui nanti.

## Search, Substitute, dan Replace

Melakukan pencarian teks pada Vim begitu sederhana, kamu hanya perlu menggunakan tanda miring garis / dan disertai dengan *pattern* atau kata kunci yang kamu ingin cari.

/pattern

Misal, kamu ingin mencari bagian kode `userName`, kamu dapat menggunakan perintah /`userName` pada Vim.

Bila Vim sudah menemukan teks dengan *pattern* atau kata kunci yang kamu cari, kamu dapat menggunakan perintah n yang berarti *next* untuk melanjutkan pencarian dengan kata kunci yang sama. Sebaliknya, bila kamu ingin kembali ke teks yang ditemukan sebelumnya, kamu dapat menggunakan perintah N (*ya, “N” kapital*) untuk kembali.

Seandainya Vim tidak menemukan teks dengan *pattern* atau kata kunci yang kamu sesuaikan, Vim akan mengeluah seperti ini:

E486: Pattern `not found: pattern`

Perintah / membuat Vim mencari ke bawah dari posisi kursor saat ini. Bila sampai bawah tidak ditemukan, maka Vim akan mengulang pencarian kembali dari atas. Hal seperti ini ditandai dengan komentar dari Vim yang secara tekstual seperti ini:

```
search hit BOTTOM, continuing at TOP
```

Sekiranya kamu merasa kata kunci yang kamu cari berada di atas kursor saat ini, maka alih-alih menggunakan tanda garis miring /, kamu dapat menggunakan tanda tanya ? dan disertai dengan kata kunci yang kamu ingin cari.

Bila awalnya /userName, maka menjadi ?userName. Kebalikan dari tanda garis miring, dengan tanda tanya maka Vim akan mencari ke atas dari posisi kursor saat ini. Bila tidak ditemukan, maka akan mengulang pencarian dari bawah. Hal ini ditandai dengan komentar dari Vim yang secara tekstual seperti ini:

```
search hit TOP, continuing at BOTTOM
```



Sebagai tambahan, kamu dapat memindahkan posisi kursor ke kata manapun, lalu tekan perintah \* untuk mencari kata tersebut ke depan, atau perintah # untuk mencarinya ke belakang.

Perlu diingat, pencarian di Vim pada dasarnya menggunakan *case-sensitive*, jadi kamu perlu memperhatikan huruf kapital dan bukan. Seandainya kamu ingin menghindari *case-sensitive*, kamu dapat menggunakan perintah :set ignorecase atau :set ic versi yang lebih ringkas.

Kamu juga dapat mengembalikan fitur *case-sensitive* tersebut dengan perintah :set noignorecase atau :set noic.

Cara lain untuk menghindari *case-sensitive* pada saat pencarian di Vim adalah dengan menambahkan \c pada akhir perintah pencarian. Merujuk ke perintah pencarian sebelumnya, maka akan menjadi /username\c.

Kamu juga dapat menggunakan *regular expression* untuk kata kunci pencarian. Sebagai contoh, kamu ingin mencari tanda titik koma pada setiap akhir baris, kamu dapat menggunakan /;\$/ ; ingin mencari teks yang mengandung .com , kamu dapat menggunakan /.\.com .

Tentu kamu dapat menemukan referensi mengenai pencarian *regular expression* yang lebih lengkap pada sumber daya lain. Saya hanya memberikan beberapa contoh saja *barusan*.

Kita sudah belajar mencari teks di Vim, saatnya belajar mengganti kata kunci yang ditemukan dengan teks lain atau di Vim disebut juga dengan *substitute*.

Untuk melakukan *search and replace* di Vim, kamu dapat menggunakan perintah :s yang berarti *substitute* dan lebih lengkapnya seperti ini:

```
:s/keyword/replaceKeyword
```

Pada dasarnya seperti itu saja. Di mana keyword adalah *pattern* atau kata kunci yang hendak kamu cari, sedangkan replaceKeyword adalah teks yang mengantikannya bila *pattern* atau kata kunci ditemukan.

Seandainya kamu memiliki teks seperti di bawah dan dengan kondisi posisi kursor berada di huruf “k” baris pertama:

```
keyword keyword  
keyword keyword
```

Maka dengan perintah sebelumnya, teks yang kamu miliki akan menjadi seperti ini:

```
replaceKeyword keyword  
keyword keyword  
keyword keyword
```

Ya, hanya satu yang diganti, tahu kenapa? Karena Vim hanya akan men-*subtitue* atau me-*replace* teks yang pertama kali ditemui saja dan pada baris di mana posisi kursor saat itu berada.

Seandainya posisi kursor berada di baris kedua seperti ini:

```
keyword keyword  
keyword keyword
```

Maka dengan perintah yang sama, hasilnya akan seperti ini:

```
keyword keyword  
replaceKeyword keyword  
keyword keyword
```

Kamu dapat menambahkan */g* diakhir perintah sebelumnya untuk melakukan *subtitue* secara global. Perintah sebelumnya menjadi seperti ini:

```
:s/keyword/replaceKeyword/g
```

Dengan perintah tersebut, semua *pattern* atau kata kunci yang ditemukan akan di-*subtitue* oleh Vim, tidak hanya satu.

```
keyword keyword  
keyword keyword  
keyword keyword
```

Dengan perintah sebelumnya, maka hasilnya akan seperti ini:

```
replaceKeyword replaceKeyword  
keyword keyword  
keyword keyword
```

Kamu juga dapat menentukan jangkauan *subtitue*. Misal, kamu ingin melakukan *substitute* dari baris pertama hingga baris kedua, maka perintah ini dapat digunakan:

```
:1,2s/keyword/replaceKeyword/g
```

Tidak peduli di mana posisi kursor kamu berada, Vim tetap akan melakukan *substitute* hanya pada baris nomor 1 hingga baris nomor 2. Kira-kira seperti ini hasilnya:

```
replaceKeyword replaceKeyword  
replaceKeyword replaceKeyword  
keyword keyword
```

Umumnya, pada editor teks konvensional, kamu melakukan *find and replace* di seluruh dokumen — dari baris 1 hingga baris terakhir. Pada Vim, hal ini dapat dilakukan dengan perintah yang mirip seperti sebelumnya. Hanya saja, angka 2 diganti dengan tanda dolar \$ sebagai interpretasi baris akhir pada dokumen yang sedang dibuka.

```
:1,$s/keyword/replaceKeyword/g
```

Dengan perintah di atas, maka kamu sudah dapat menebak hasilnya. Ya, semua *pattern* di semua baris akan di-*substitute*.

Perintah 1,\$ dapat disingkat dengan tanda persen %. Menjadi seperti ini:

```
:%s/keyword/replaceKeyword/g
```

Sama seperti fitur pencarian Vim sebelumnya, *substitute* juga menggunakan *case-sensitive*. Kamu dapat menggunakan cara yang sama seperti di fitur pencarian untuk mengabaikan *case-sensitive*. Tambahan, kamu dapat menggunakan perintah i di akhir perintah untuk mengabaikan *case-sensitive* pada fitur *substitute*.

```
:%s/keyword/replaceKeyword/gi
```

Selain itu, pada bagian *pattern* juga kamu dapat menulis *regular expression* untuk pencarian yang lebih lanjut.

Sejurnya, *substitute* adalah fitur yang jarang sekali saya gunakan. Saya lebih memilih menggunakan fitur pencarian menggunakan

perintah / atau ?. Lalu saya menggantinya secara manual dengan menggunakan perintah c atau d bila saya ingin menghapusnya.

Alasan saya melakukan ini karena saya ingin meninjau satu per satu teks atau kode yang hendak saya ganti atau hapus. Agak berisiko bagi saya menggunakan *substitute* pada basis kode yang sudah besar. Tentu saja hal ini tergantung pada kasus~

Terakhir, sebaiknya kita perlu memulai membiasakan istilah *substitute* pada fitur ini. Karena istilah *replace* seringkali merujuk pada fitur Vim yang lain dengan perintah yang berbeda juga. Kamu dapat menggunakan perintah `r{char}` untuk me-*replace* satu karakter yang sedang berada di bawah kursor.

Sebagai contoh kamu memiliki teks seperti ini:

Aku

Kemudian kamu gunakan perintah `rs`. Maka hasilnya seperti ini:

sku

Karakter A di-*replace* dengan s.

Kamu juga dapat me-*replace* lebih dari satu karakter dengan perintah R . Perintah tersebut akan membawa kamu ke mode *replace*. Pada mode tersebut, setiap karakter yang kamu ketik akan menggantikan karakter yang berada di bawah posisi kursor saat itu sampai kamu kembali lagi ke mode normal dengan tombol ESC.

Mode ini yang hampir tidak pernah saya gunakan. Namun perintah r sangat sering saya gunakan.

## Visual Mode

Pada editor teks konvensional kamu dapat menggunakan *mouse* untuk menyeleksi teks atau kode. Sebagai alternatif, kamu juga dapat menggunakan tombol SHIFT disertai arah panah.

Di Vim, untuk menyeleksi teks, kamu dapat menggunakan mode *visual*. Pada mode *visual*, *motion* akan bertindak menyeleksi sesuai dengan perintahnya, dan *operator* tidak perlu lagi menunggu *motion* untuk bertindak.

Perintah `v` untuk masuk ke mode *visual*. Saat berada di mode *visual* kamu dapat menggunakan *motion* seperti `w`, `}` dan sebagainya untuk mulai menyeleksi teks. Mode *visual* ditandai dengan teks “-- VISUAL --” di kiri-bawah jendela *terminal emulator*.

A screenshot of the Alacritty terminal window titled "Alacritty > vim". The terminal displays a snippet of JavaScript code:

```
function add(a, b) {  
    return a + b;  
}  
  
console.log(add(1, 2));
```

The text "console.log(add(1, 2));" is selected. At the bottom left of the terminal window, the text "-- VISUAL --" is visible. The status bar at the bottom shows the current line number "5", the first character of the line "5,1", and the word "All".

Beralih ke mode *visual*

Saat ini warna latar belakang dan warna teks seleksi pada mode *visual* tidak begitu kontras, sehingga teks yang diseleksi tidak begitu terlihat jelas.

A screenshot of an Alacritty terminal window. The title bar says "Alacritty > vim". The terminal shows a snippet of JavaScript code:function add(a, b) {  
 return a + b;  
}  
  
console.log(add(1, 2));The text "VIM" is highlighted in blue. The status bar at the bottom right shows "5", "5,12", and "All".

Warna seleksi tidak kontras

Bila kamu mengalami hal yang sama, kita dapat mengubah warna seleksi dengan perintah:

```
:highlight Visual ctermbg=yellow ctermfg=black
```

Sekarang seharusnya warna seleksi pada mode *visual* akan lebih kontras.

A screenshot of an Alacritty terminal window. The title bar says "Alacritty > vim". The terminal shows the same snippet of JavaScript code. The text "VIM" is now highlighted in orange. The status bar at the bottom right shows "5", "5,11", and "All".

Warna seleksi lebih kontras

Sekarang kita akan mencoba menyeleksi satu "paragraf" ke depan dan menghapusnya, gunakan perintah `v}d`. Dengan catatan posisi kursor berada pada karakter pertama di awal paragraf. Penjelasannya seperti ini:

- `v` masuk ke mode *visual*
- `}` memindahkan kursor satu "paragraf" ke depan
- `d` menghapusnya

Sebagai contoh, terdapat kode JavaScript berikut dan posisi kursor berada di karakter `f` dalam kode function.

```
function add(a, b) {  
    return a + b;  
}  
  
console.log(add(1, 2));
```

Jika perintah `v}d` diberikan, maka Vim akan menghapus bagian kode mulai dari `f` hingga spasi kosong di bawah tanda `}``.

Pada mode normal, *operator* seperti perintah `d` akan menunggu *motion* untuk melakukan tugasnya, misal `dw` untuk *delete word*. Tetapi, pada mode *visual*, tidak perlu lagi menunggu *motion*. Karena *operator* tersebut (dalam hal ini adalah `d`) sudah tahu teks yang perlu dihapus, yaitu teks yang sedang berada dalam cakupan seleksi.

Selain menggunakan *operator* `d` untuk menghapus, kamu juga dapat menggunakan *operator* lainnya pada mode *visual*:

- `c` untuk menghapus dan masuk ke mode *insert*
- `y` untuk *yank* (menyalin/copy)
- `>` untuk indentasi

Di samping mode *visual*, terdapat juga mode *visual line*. Mirip seperti mode *visual*, tapi pada mode *visual line* hanya dapat menyeleksi per

baris saja. Tidak peduli posisi kursor kamu berada di awal, di tengah, atau di akhir baris, ia akan tetap menyeleksi satu baris tersebut.

Perintah **V** untuk masuk ke mode visual line. Mode *visual line* ditandai dengan teks “**-- VISUAL LINE --**” di kiri-bawah jendela *terminal emulator*.

A screenshot of the Alacritty terminal window titled "Alacritty > vim". The terminal displays the following JavaScript code:

```
function add(a, b) {
    return a + b;
}

console.log(add(1, 2));
```

Below the code, there are approximately 20 horizontal tilde (~) characters. At the bottom of the terminal window, the text "-- VISUAL LINE --" is displayed. The status bar at the bottom shows the line numbers 1, 11, and All.

Beralih ke mode visual line

Mode ini berguna ketika kamu memang ingin menghapus satu blok kode namun posisi kursor kamu tidak berada di awal blok kode tersebut.

Selain kedua mode tersebut, terdapat juga mode *visual block*. Mode ini agak sulit dijelaskan, lebih baik kita langsung coba saja.

Misal, kita punya kode JavaScript seperti ini:

```
function welcome(name) {
    return `Welcome, ${name}`;
}
```

Katakanlah kita tidak ingin menggunakan lagi, kita dapat memberikan komentar seperti ini:

```
// function welcome(name) {  
//   return `Welcome, ${name}`;  
// }
```

Ya, kita ingin memberikan kode komentar // pada setiap baris kode JavaScript tersebut. Dengan mode *visual block* kita tidak perlu memasukkan komentar tersebut satu per satu.

Pertama, posisikan kursor pada awal baris di karakter f pada kata `function`, masuk ke mode *visual block* dengan tombol kombinasi CTRL-v, pindahkan kursor ke bawah sebanyak dua kali sampai kursor berada di tanda `}``. Kita dapat menggunakan perintah `2j` agar lebih cepat.

Saat ini *visual block* sedang menyeleksi 3 baris, sekarang kita dapat gunakan perintah `I` untuk masuk ke mode *insert* dengan posisi kursor berada pada awal baris. Pada mode *insert* inilah kita masukkan kode komentar `//`. Terakhir kembali lagi ke mode normal.

Seharusnya hasilnya seperti ini:

```
// function welcome(name) {  
//   return `Welcome, ${name}`;  
// }
```

Untuk menghapus komentar, kita dapat mengulang urutan perintah yang mirip seperti sebelumnya. Posisikan kursor pada awal baris di karakter f pada kata `function`, masuk ke mode *visual block* dengan tombol kombinasi CTRL-v, pindahkan kursor ke bawah sebanyak dua kali sampai kursor berada di tanda `}``. Kita dapat menggunakan perintah `2j` agar lebih cepat.

Saat ini *visual block* sedang menyeleksi 3 baris, sekarang geser posisi kursor ke kanan sebanyak 2 kali, kita dapat menggunakan perintah `2l`. Terakhir kita dapat hapus teks yang sedang diseleksi dengan perintah `d`. Maka saat ini kode komentar sudah hilang.

Itulah kasus yang paling sering saya gunakan pada mode *visual block*. Pada dasarnya ketiga mode *visual* ini sama saja, yang membedakan adalah bagaimana cara Vim menyeleksi teks pada masing-masing mode *visual*.

## Lompat-Melompat

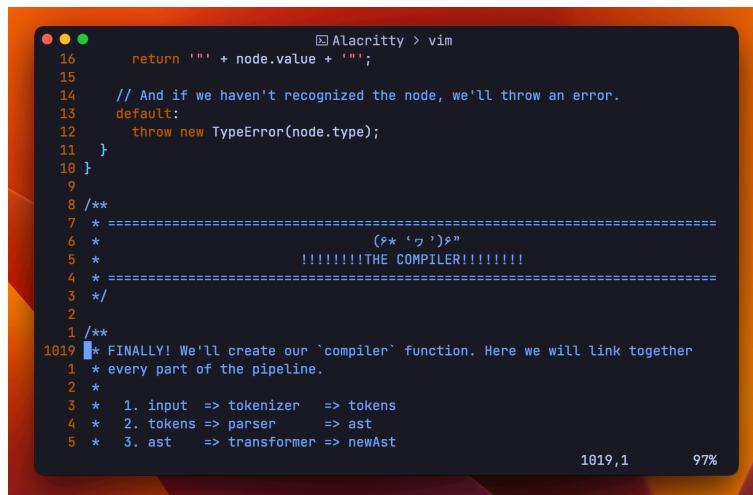
Kamu dapat memindahkan posisi kursor ke suatu tempat dengan cepat, seperti pindah ke atas berkas, ke bawah berkas, atau pergi ke spesifik nomor baris. Selain itu kamu juga dapat men-*scroll* layar Vim ke atas atau ke bawah. Semua itu dilakukan hanya dengan *keyboard*.

Berikut ini saya rangkum perintah-perintahnya:

- gg untuk memindahkan kursor ke atas dokumen
- G untuk memindahkan kursor ke bawah dokumen
- :n di mana n adalah angka — untuk lompat ke spesifik baris
- nGG di mana n adalah angka — sama seperti :n
- CTRL-y untuk men-*scroll* Vim satu baris ke atas
- CTRL-e untuk men-*scroll* Vim satu baris ke bawah
- CTRL-u untuk memindahkan kursor dan layar  $\frac{1}{2}$  halaman
- CTRL-d untuk memindahkan kursor dan layar  $\frac{1}{2}$  halaman

Agar lebih cepat lagi, biasanya pengguna Vim menggunakan *relative number*. Di mana dengan *relative number*, nomor baris di atas dan di bawah kursor akan selalu 1, 2, 3 dan seterusnya.

Fitur ini sangat berguna, karena kita memiliki referensi nomor yang bisa digunakan untuk melompat ke atas dan ke bawah. Untuk mengaktifkan fitur *relative number*, kita dapat menggunakan perintah :set relativenumber dan perintah :set number untuk menampilkan nomor baris.



```
 16     return '"' + node.value + '"';
15
14     // And if we haven't recognized the node, we'll throw an error.
13     default:
12         throw new TypeError(node.type);
11     }
10 }
9 /**
8 */
7 * =====
6 *          (＊'＊)♪
5 *          !!!!!!!THE COMPILER!!!!!!
4 * =====
3 */
2
1 /**
1019 * FINALLY! We'll create our `compiler` function. Here we will link together
1 * every part of the pipeline.
2 *
3 *   1. input  -> tokenizer  -> tokens
4 *   2. tokens -> parser    -> ast
5 *   3. ast    -> transformer -> newAst
1019,1      97%
```

Mengaktifkan fitur *relative number* dan nomor baris

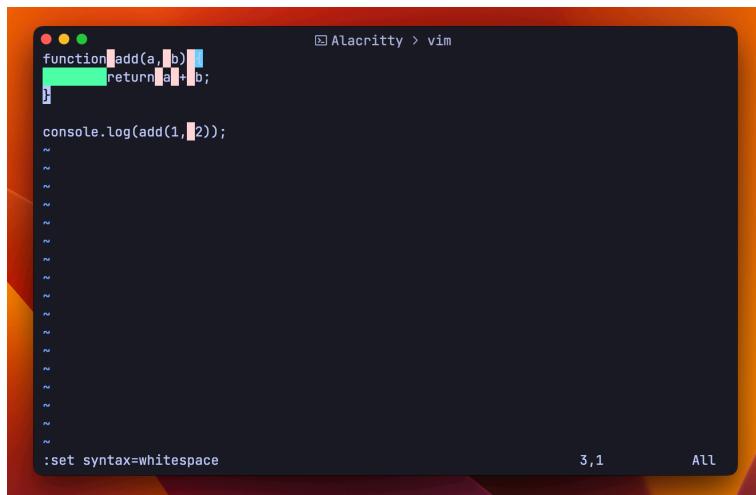
Lihat kembali kode pada tangkapan layar di atas. Sebagai contoh, saya ingin melompat ke kode `throw new TypeError(node.type);`, saya dapat menggunakan perintah `12k` untuk mencapainya. Karena kode tersebut berada di nomor baris 12 di atas posisi kursor saat ini. Jadi, perintah `12k` berarti memindahkan kursor ke atas sebanyak 12 kali.

Karena nomor baris saat ini relatif, maka setiap posisi kursor berpindah ke atas atau ke bawah, referensi nomor baris akan berubah-ubah.

Dengan seperti ini, kita akan semakin lebih cepat memindahkan posisi kursor ke atas atau ke bawah.

## Indentasi

Pada Vim yang saya gunakan, untuk indentasinya menggunakan karakter *tab* ketimbang spasi. Lebar *tab*-nya setara dengan 8 spasi. Untuk mengetahui informasi ini, kamu dapat menggunakan perintah `:set syntax=whitespace`.

A screenshot of a terminal window titled "Alacritty > vim". The terminal displays the following code:

```
function add(a, b)
    return a + b;
}

console.log(add(1, 2));
```

The code uses Vim's whitespace syntax highlighting, where tabs are shown as green horizontal bars and spaces as red vertical bars. The status bar at the bottom shows the command ":set syntax=whitespace". The bottom right corner of the terminal window indicates "3,1" and "All".

Mengubah jenis sintaksis menjadi *whitespace*

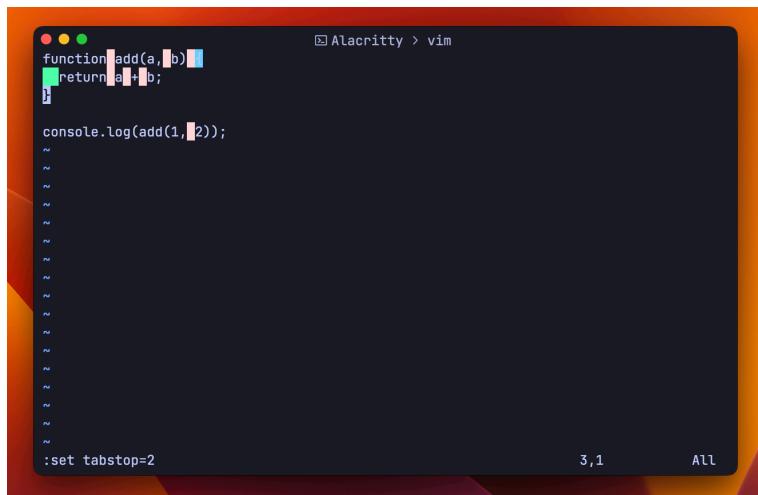
Ini adalah aturan *syntax highlighting* untuk bahasa pemrograman Whitespace<sup>11</sup>. *Tab* ditampilkan dengan warna hijau, sedangkan spasi ditampilkan dengan warna merah.

Tentu saja angka 8 adalah angka yang sangat lebar untuk ukuran indentasi. Umumnya, orang-orang menggunakan ukuran 4 atau 2. Dan sampai hari ini masih tetap menjadi perdebatan antara menggunakan spasi atau *tab* untuk karakter indentasi.

Secara bawaan, seperti yang saya tampilkan di atas, Vim menggunakan *tab* ketimbang spasi. Apabila kamu sudah terbiasa dengan *tab* dan hendak mengubah ukurannya saja, maka kamu dapat mengaturnya dengan perintah :set tabstop=n yang di mana n adalah lebar dari *tab*.

---

<sup>11</sup> "Whitespace (programming language)" [https://en.wikipedia.org/wiki/Whitespace\\_%28programming\\_language%29](https://en.wikipedia.org/wiki/Whitespace_%28programming_language%29)

A screenshot of a terminal window titled "Alacritty > vim". The terminal displays a snippet of JavaScript code:

```
function add(a, b){  
    return a + b;  
}  
  
console.log(add(1, 2));
```

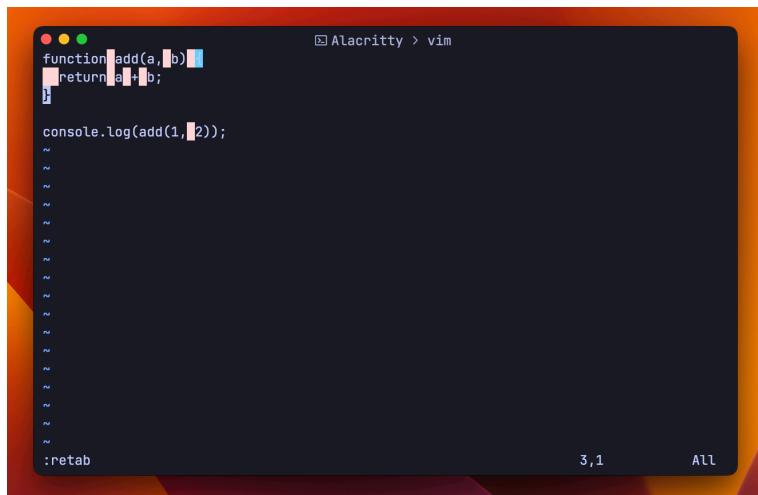
The code uses tabs for indentation. At the bottom of the screen, the status bar shows the command ":set tabstop=2" and the coordinates "3,1" and "All".

Kode dengan ukuran tab 2

Seandainya kamu satu selera dengan saya dan ingin mengubah dari *tab* menjadi spasi, maka kamu dapat mengurnya dengan perintah :*set expandtab*.

- Konfigurasi *tabstop=n* akan mengatur lebar dari karakter *tab*.
- Konfigurasi *expandtab* akan mengubah karakter *tab* menjadi spasi

Walaupun ukuran indentasinya sudah disesuaikan, namun karakter saat ini yang masih digunakan adalah *tab* bukan spasi. Untuk mengubahnya, kita dapat menggunakan perintah :*retab*.

A screenshot of a terminal window titled "Alacritty > vim". The terminal shows a piece of JavaScript code:

```
function add(a, b) {
    return a + b;
}

console.log(add(1, 2));
```

The code uses tabs for indentation. In the bottom left corner of the terminal, the command ":retab" is visible. In the bottom right corner, the status bar shows "3,1" and "All".

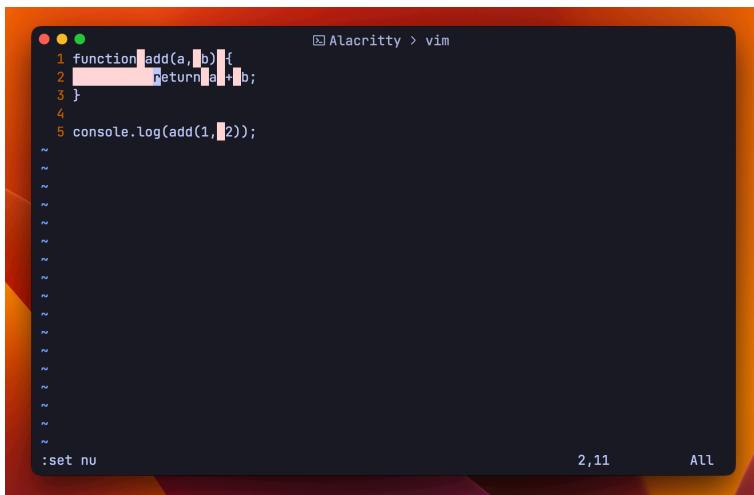
Mengubah tab menjadi spasi

Karena kita menggunakan konfigurasi `expandtab`, jadi kita perlu menggunakan konfigurasi `softtabstop=n` ketimbang `tabstop=n`. Karena ketika kita menekan tombol TAB, Vim akan memasukkan jumlah karakter spasi sesuai dengan jumlah `softtabstop`.

Untuk melakukan indentasi manual, kita dapat menggunakan *operator* `>`, `<` dan `=`.

- `>` untuk menambah indentasi
- `<` untuk mengurangi indentasi
- `=` untuk memperbaiki indentasi

Sebagai contoh, kita dapat menggunakan perintah `>i{` untuk menambah indentasi semua kode yang berada di dalam tanda kurung kurawal `{ ... }`; atau menggunakan perintah `>>` untuk menambah indentasi satu baris pada posisi kursor berada; atau menggunakan mode *visual*.



A screenshot of a terminal window titled "Alacritty > vim". The terminal displays the following JavaScript code:

```
function add(a, b){  
    return a + b;  
}  
  
console.log(add(1, 2));
```

The cursor is positioned at the start of the second line. Below the code, the status bar shows ":set nu" on the left, "2,11" in the center, and "All" on the right.

Memberikan indentasi pada baris kedua dengan perintah >>

Indentasi yang diberikan tetap terlalu lebar, tidak sesuai dengan konfigurasi yang kita berikan sebelumnya. Apakah konfigurasinya tidak terbaca? Tentu bukan. Konfigurasi untuk mengatur lebar indentasi pada saat menekan tombol TAB dan pada saat menggunakan perintah >, <, atau = itu berbeda konfigurasinya.

Sebelumnya kita hanya mengatur lebar indentasi pada saat menekan tombol TAB saja. Untuk mengatur lebar indentasi pada saat menggunakan perintah >, <, atau =, kita dapat menggunakan konfigurasi `shiftwidth=n`.

Konfigurasi terakhir untuk saat ini prihal indentasi adalah *autoindent* dan *smartindent*. Kedua konfigurasi ini penting dan membantu. Tanpa konfigurasi ini, kita perlu memberikan indentasi secara manual dengan menekan tombol TAB setiap kali menulis kode pada baris baru.

Misal, kita memiliki kode JavaScript berikut dan posisi kursor berada di tanda kurawal }.

```
function add(a, b) {  
    return a + b;  
}
```

Jika kita ingin membuat baris dengan menggunakan perintah o pada mode normal, maka kursor akan berada di baris baru namun tanpa indentasi yang sama dengan bagian kode `return` ....

```
function add(a, b) {  
    |  
    return a + b;  
}
```

Posisi kursor akan lebih memudahkan bila memiliki indentasi sama seperti bagian kode `return`.

```
function add(a, b) {  
    |  
    return a + b;  
}
```

Pada beberapa kasus mungkin fitur ini sudah otomatis aktif, namun, bila seandainya belum aktif hal seperti di atas akan terjadi. Kita dapat mengaktifkan fitur *autoindent* dengan perintah `:set autoindent` atau `:set ai` versi yang lebih ringkas. Konfigurasi ini akan memberitahu Vim untuk menerapkan indentasi dari baris saat ini ke baris berikutnya (atas dan bawah; dengan menekan tombol `ENTER`, o atau O).

Sedangkan untuk mengaktifkan fitur *smartindent*, kita dapat menggunakan perintah `:set smartindent` atau `:set si` untuk versi yang lebih ringkas. Seperti namanya *smart*, fitur ini akan menyesuaikan gaya indentasi sesuai dengan jenis sintaksis yang kita sedang gunakan.

Kita juga dapat menyeleksi semua kode dan menggunakan *operator =* untuk menyesuaikan indentasi dengan konfigurasi saat ini.

Untuk menyeleksi semua kode, pindahkan kursor ke paling atas, beralih ke mode *visual*, pindahkan kursor ke paling bawah, lalu beri perintah =. Bila digabungkan, maka perintahnya seperti ini ggvG=.

## Konfigurasi

Vim memungkinkan kita untuk menulis konfigurasi sendiri untuk mengatur prilaku Vim. Konfigurasi tersebut bernama vimrc. Vi menggunakan nama exrc, sedangkan vimrc adalah nama spesifik dari Vim.

Kita dapat menulis berkas konfigurasi ini di dalam folder \$HOME/.vimrc atau \$HOME/.vim/vimrc, bila kita menggunakan Unix atau Unix-*like* seperti macOS atau Linux. Untuk Windows, berkas konfigurasi dapat ditaruh di folder \$HOME/\_vimrc atau \$HOME/vim-files/vimrc.

Saya menggunakan macOS, jadi saya akan menulis konfigurasi vimrc di dalam folder \$HOME/.vimrc atau ~/.vimrc.

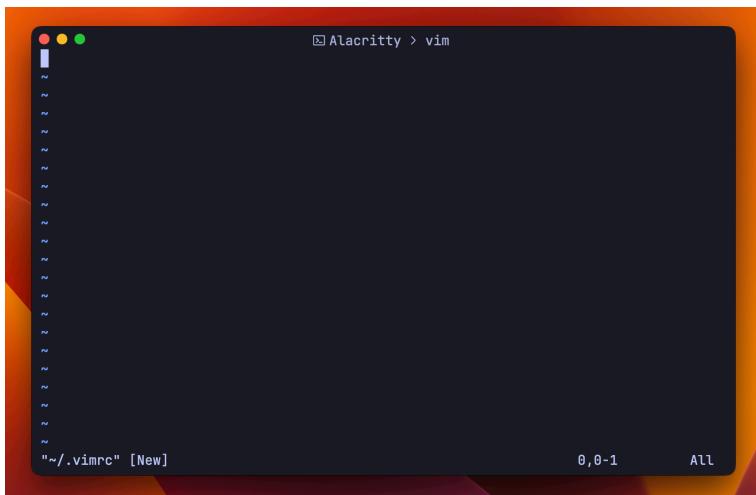
Untuk membuat berkas konfigurasi baru di dalam Vim, kita dapat menggunakan perintah :e ~/.vimrc Unix dan Unix-*like*, dan :e ~/\_.vimrc untuk Windows. Di mana tanda ~ adalah alias untuk \$HOME, untuk mengetahui nilai \$HOME, kita dapat menggunakan perintah :echo \$HOME di Vim, hasilnya kira-kira seperti ini:

```
/Users/mhdnaualazhar
```

Di Windows:

```
C:\Users\mhdnaualazhar
```

Vim seharusnya sedang menampilkan berkas baru kosong.



Berkas konfigurasi baru di Vim

Konfigurasi pada berkas *vimrc* ditulis dengan bahasa Vimscript, walaupun terdapat sebagian orang membenci bahasa ini. Berbeda dengan Vim, Neovim memungkinkan kamu menulis konfigurasi dengan bahasa Lua. Untuk saat ini, karena kita menggunakan Vim, kita akan menulisnya dengan Vimscript.

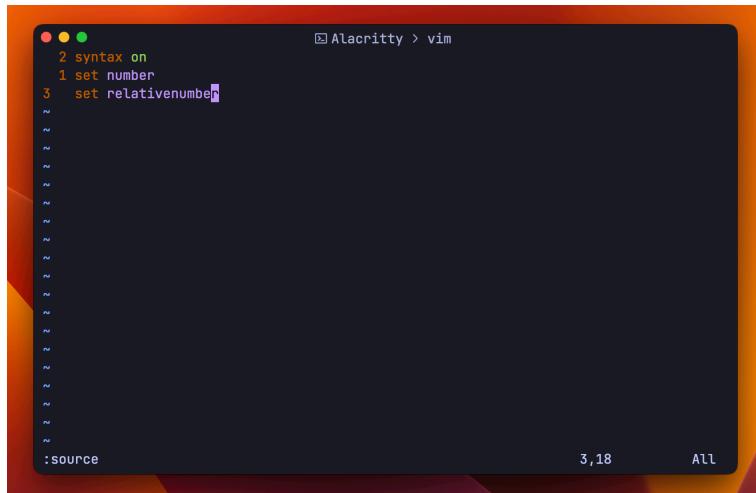
Di dalam berkas *vimrc* kita hanya akan menulis perintah-perintah yang sebelumnya pernah kita gunakan, seperti `:syntax on`, `:set number` dan sebagainya. Perintah-perintah tersebut sebetulnya konfigurasi, namun tidak *persistent* — hanya untuk sementara.

Kita dapat menulis kode seperti ini di dalam berkas *vimrc* untuk menghidupkan fitur *syntax highlighting*, *line numbers*, dan *relative number*.

```
syntax on  
set number  
set relativenumber
```

Lalu simpan berkas tersebut dengan perintah `:w`.

Ketika kita melakukan perubahan pada berkas konfigurasi *vimrc*, kita perlu memberitahu Vim untuk memuat ulang konfigurasi tersebut dengan perintah `:source`.



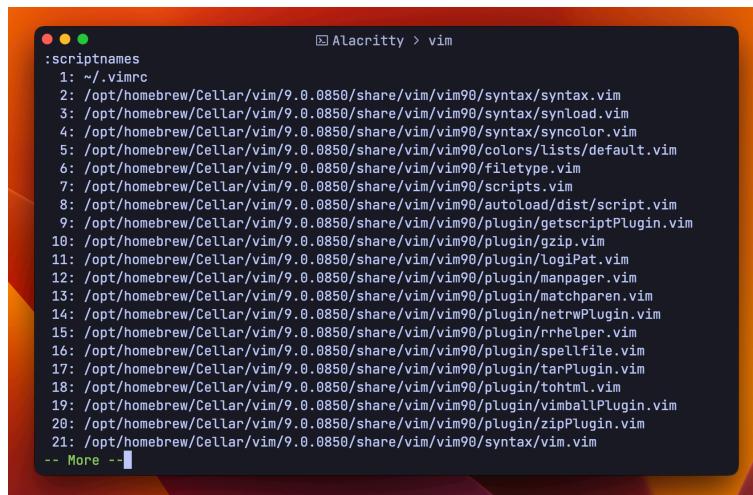
The screenshot shows a terminal window titled "Alacritty > vim". Inside the terminal, the command `:source` is being typed at the bottom. Above it, there is configuration text starting with `2 syntax on`. The status bar at the bottom right shows the current position as `3,18` and the buffer name as `All`.

Memuat ulang konfigurasi Vim

Perintah tersebut digunakan apabila kita ingin memuat ulang konfigurasi tanpa keluar Vim, pada dasarnya Vim akan selalu memuat konfigurasi terbaru ketika dimulai – dalam kata lain, kita dapat keluar Vim dan membukanya lagi untuk membuka Vim dengan konfigurasi terbaru.

Kita dapat mengetahui informasi mengenai daftar konfigurasi atau *script* yang dibaca oleh Vim saat ini dengan perintah:

```
:scriptnames
```

A screenshot of a terminal window titled "Alacritty > vim". The window contains a list of file paths under the command ":scriptnames". The list includes: 1: ~/vimrc, 2: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/syntax/syntax.vim, 3: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/syntax/synload.vim, 4: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/syntax/syncolor.vim, 5: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/colors/lists/default.vim, 6: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/filetype.vim, 7: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/scripts.vim, 8: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/autoload/dist/script.vim, 9: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/getscriptPlugin.vim, 10: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/gzip.vim, 11: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/logipat.vim, 12: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/manpager.vim, 13: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/matchparen.vim, 14: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/netrwPlugin.vim, 15: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/rrhelper.vim, 16: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/spellfile.vim, 17: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/tarPlugin.vim, 18: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/tohtml.vim, 19: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/vimballPlugin.vim, 20: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/zipPlugin.vim, 21: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/syntax/vim.vim. A green status bar at the bottom right shows "-- More --".

```
:scriptnames
1: ~/vimrc
2: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/syntax/syntax.vim
3: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/syntax/synload.vim
4: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/syntax/syncolor.vim
5: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/colors/lists/default.vim
6: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/filetype.vim
7: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/scripts.vim
8: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/autoload/dist/script.vim
9: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/getscriptPlugin.vim
10: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/gzip.vim
11: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/logipat.vim
12: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/manpager.vim
13: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/matchparen.vim
14: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/netrwPlugin.vim
15: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/rrhelper.vim
16: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/spellfile.vim
17: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/tarPlugin.vim
18: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/tohtml.vim
19: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/vimballPlugin.vim
20: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/plugin/zipPlugin.vim
21: /opt/homebrew/Cellar/vim/9.0.0850/share/vim/vim90/syntax/vim.vim
-- More --
```

Daftar konfigurasi Vim

Kita bisa perhatikan, dalam kasus saya, berkas *vimrc* berada diurutan pertama. Sisanya adalah *plugin-plugin* yang dimuat oleh Vim secara bawaan. Jika kamu tidak menemukannya, kamu dapat mengeluarkan Vim dan membukanya kembali dan ulangi perintah sebelumnya.

Selain konfigurasi tadi, kita dapat menulis konfigurasi indentasi yang kita sudah lakukan sebelumnya.

```
set expandtab
set shiftwidth=2
set softtabstop=2
set autoindent
set smartindent
```

Kamu juga dapat mengatur konfigurasi *color scheme* di *vimrc*, misal:

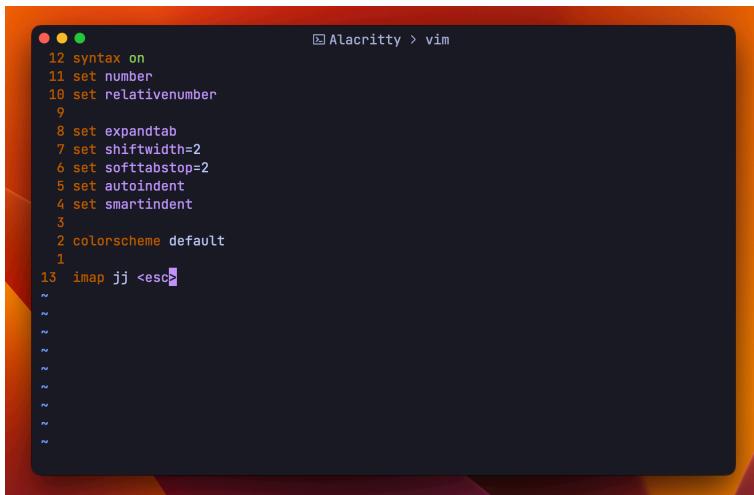
```
colorscheme habamax
```

Umumnya, pengguna Vim (termasuk saya) membuat pemetaan tombol **ESC** ke tombol **jj**. Ini bertujuan agar mempermudah ketika hendak beralih dari mode *insert* ke mode normal. Karena tombol **ESC** terlalu jauh untuk dijangkau. Bila kamu tertarik juga, kamu dapat melakukannya seperti ini:

```
imap jj <esc>
```

Ketika kamu berada di dalam mode *insert* dan hendak kembali ke mode normal, kamu hanya perlu menekan tombol jj, tidak perlu jauh-jauh ke tombol ESC.

Sampai sejauh ini, konfigurasi yang kita tulis sudah bisa digunakan sebagai modal awal untuk mulai beralih ke Vim.

A screenshot of a terminal window titled "Alacritty > vim". The window contains the contents of a vimrc file. The text is color-coded: numbers are orange, keywords like "syntax", "set", and "imap" are red, and other text is white. The configuration includes syntax highlighting, number and relative numbering, expandtab, shiftwidth, softtabstop, autoindent, smartindent, colorscheme default, and the key mapping "imap jj <esc>".

```
12 syntax on
11 set number
10 set relativenumber
9
8 set expandtab
7 set shiftwidth=2
6 set softtabstop=2
5 set autoindent
4 set smartindent
3
2 colorscheme default
1
13 imap jj <esc>
```

Keseluruhan konfigurasi sejauh ini

Tentu untuk menyetarakan Vim dengan VS Code, kita perlu menulis konfigurasi yang lebih panjang lagi, serta membutuhkan beberapa *plugin-plugin* tambahan.

## Plugin

Vim memiliki ekosistem *plugin* yang memungkinkan kamu memperluas fungsionalitas Vim. Di Vim, *plugin* tidak lebih dari sekadar sebuah Vimscript yang diproses secara otomatis pada saat Vim dimulai.

Kamu dapat menggunakan perintah `:scriptnames` untuk melihat daftar *script* yang dieksekusi oleh Vim saat ini. Umumnya, Vim akan membaca terlebih dahulu konfigurasi yang kita tulis (di `~/.vimrc`)

atau `~/_vimrc`), dan berikutnya memproses beberapa *plugin*-*plugin* bawaan Vim.

Pada versi 8 atau lebih tinggi, Vim memiliki *plugin manager* bawaan bernama packages. Sederhananya ini hanyalah sebuah folder yang dapat kita gunakan menaruh *plugin*-*plugin* yang kita inginkan. Semua *plugin* tersebut akan diproses otomatis saat Vim dimulai.

Jadi selain memproses berkas konfigurasi *vimrc*, dan *plugin*-*plugin* bawaan, Vim juga akan memproses *script* atau *plugin*-*plugin* yang berada di dalam folder `~/.vim/pack/*/start` pada Unix atau Unix-like (`~/vimfiles/pack/*/start` untuk Windows) secara otomatis pada saat Vim dimulai.

Buatlah folder dengan nama *pack* pada lokasi seperti ini:

```
~/vim/pack
```

Untuk Windows:

```
~/vimfiles/pack
```

Struktur folder untuk *plugin* yang direkomendasikan oleh Vim adalah seperti ini:

```
~/vim/pack/namaplugin/start/namaplugin
```

Untuk Windows:

```
~/vimfiles/pack/namaplugin/start/namaplugin
```

Sebagai contoh, sekarang kita akan mencoba untuk memasang *plugin* bernama *emmet-vim* (<https://github.com/mattn/emmet-vim>).

Dengan *plugin* tersebut, kita dapat menulis kode HTML dengan sintaksis yang terinspirasi dari CSS.

Misal, kamu menulis singkatan kode berikut:

```
ul>li*5
```

Dengan Emmet, kamu dapat mengkonversi kode tersebut menjadi:

```
<ul>
  <li></li>
  <li></li>
  <li></li>
  <li></li>
  <li></li>
</ul>
```

VS Code secara bawaan sudah memiliki *plugin* ini terpasang, apabila kamu belum tahu.

Untuk memasang *plugin* Emmet di Vim, kita perlu mengunduh *repository* dari *plugin* tersebut. Untuk itu, kita dapat menggunakan Git atau mengunduh berkas Zip-nya. Sebelum, melakukan kloning, kita perlu membuat folder untuk menaruh *plugin* Emmet.

Buatlah folder baru bernama *start* di dalam folder baru bernama *emmet*, seperti ini:

```
~/.vim/pack/emmet/start
```

Untuk Windows:

```
~/vimfiles/pack/emmet/start
```

Sekarang saatnya *clone repository* dengan Git:

```
git clone --depth 1 https://github.com/mattn/emmet-vim ~/.vim/pack/emmet/start/emmet
```

Atau unduh berkas Zip-nya di Github:

<https://github.com/mattn/emmet-vim>

Lalu ekstrak berkas tersebut dan akan ada folder benama *emmet-vim-master*, pindahkan folder tersebut ke dalam folder `~/.vim/pack/emmet/start` (`~/vimfiles/pack/emmet/start` untuk Windows) dan

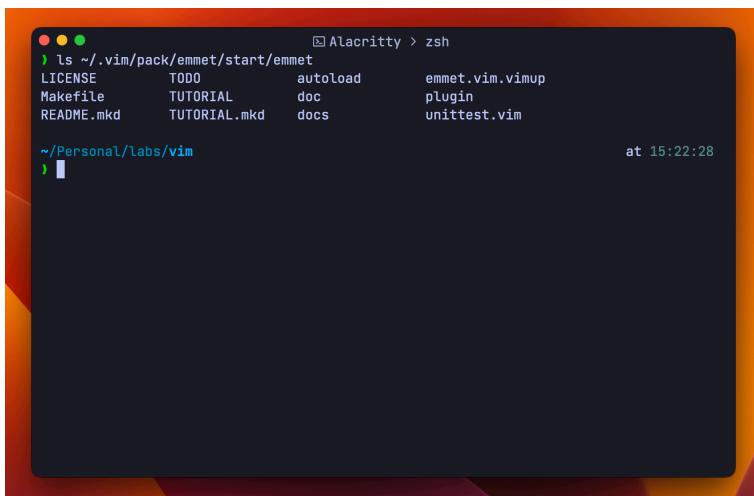
ubah nama folder tersebut dari *emmet-vim-master* menjadi *emmet*, seharusnya seperti ini:

```
~/vim/pack/emmet/start/emmet
```

Atau untuk Windows:

```
~/vimfiles/pack/emmet/start/emmet
```

Di dalam folder tersebut terdapat banyak berkas dan folder.



```
ls ~/vim/pack/emmet/start/emmet
LICENSE      TODO      autoload      emmet.vim.vimup
Makefile     TUTORIAL  doc          plugin
README.mkd   TUTORIAL.mkd  docs        unittest.vim

~/Personal/labs/vim                                at 15:22:28
```

Isi folder `~/vim/pack/emmet/start/emmet`

Sekarang kita dapat membuka Vim, apabila kamu sedang membukanya, lebih baik menutup dan membuka Vim kembali.

Untuk mencobanya, kita dapat menggunakan kode singkatan berikut:

```
html:5
```

Pastikan posisi kursor kamu berada di akhir singkatan, dalam contoh ini adalah angka 5. Kemudian gunakan tombol kombinasi **CTRL-y** dan disambung dengan tombol **,**.

Hasilnya akan seperti ini:

```
<!DOCTYPE html>
<html lang="en">
```

```
<head>
  <meta charset="UTF-8">
  <title></title>
</head>
<body>

</body>
</html>
```

Kamu dapat memeriksa dokumentasi Emmet<sup>12</sup> untuk mengeksplor singkatan-singkatan yang lain.

Kembali lagi ke konsep *plugin* di Vim.

Sekarang kita sudah berhasil memasang satu *plugin* di Vim dan menggunakan struktur folder yang direkomendasikan oleh Vim. Sebenarnya kita bisa menyederhanakan struktur folder untuk menaruh *plugin-plugin* pihak ketiga yang kita pasang. Seperti ini misalnya:

```
~/.vim/pack/vendor/start
```

Untuk Windows:

```
~/vimfiles/pack/vendor/start
```

Kita dapat menaruh *plugin-plugin* di dalam folder tersebut. Istilah "vendor" digunakan sebagai indikator bahwa folder tersebut berisi *plugin-plugin* pihak ketiga yang bukan ditulis olehmu.

Selain itu, seandainya kita tidak ingin sebuah *plugin* diproses secara otomatis oleh Vim saat Vim dibuka, kita dapat mengubah nama folder *start* menjadi *opt*. Untuk mengaktifkannya, kita dapat menggunakan perintah :packadd.

Misal, sebelumnya kita memiliki folder untuk *plugin* Emmet seperti ini:

---

<sup>12</sup> "Emmet — the essential toolkit for web-developers" <https://docs.emmet.io/>

```
~/.vim/pack/emmet/start/emmet
```

Untuk Windows:

```
~/vimfiles/pack/emmet/start/emmet
```

Kita dapat mengubah namanya menjadi:

```
~/.vim/pack/emmet/opt/emmet
```

Untuk Windows:

```
~/vimfiles/pack/emmet/opt/emmet
```

Sekarang *plugin* tersebut tidak diproses oleh Vim secara otomatis. Kita dapat menggunakan perintah :packadd emmet untuk mengaktifkannya.

Begitulah cara yang paling dasar mengelola *plugin* di Vim. Umumnya pengguna Vim menggunakan *package manager* pihak ketiga untuk mengelola *plugin* yang lebih mudah, seperti:

- vim-plug (<https://github.com/junegunn/vim-plug>)
- pathogen.vim (<https://github.com/tpope/vim-pathogen>)
- vundle.vim (<https://github.com/VundleVim/Vundle.vim>)

Setidaknya ketiga itu yang sering kedengaran namanya. Selain itu, mungkin kamu dapat bereksplorasi lagi.

Selain *plugin* Emmet yang kita pasang tadi, terdapat banyak sekali *plugin-plugin* Vim yang dapat kita pasang. Mulai yang memiliki fungsionalitas, atau hanya sebagai estetika saja.

- *fzf.vim*: sebuah *wrapper* Vim untuk *command-line fuzzy finder*
- *NERDTree*: sebuah *plugin* untuk *tree eksplorer*
- *vim-airline*: sebuah *plugin* untuk menampilkan baris status dan baris *tab*
- *coc.nvim*: sebuah *autocompletion* dan *language server* untuk Vim dan Neovim

Kamu dapat mengunjungi Vim Awesome<sup>13</sup> untuk mendapatkan daftar yang lebih lengkap.

## Buffer

Saat kita membuka berkas di Vim, isi berkas tersebut dibaca ke dalam sebuah *buffer*. Di Vim, *buffer* adalah sebuah area memori yang digunakan untuk menyimpan sementara isi berkas. Penyuntingan di Vim akan membuat perubahan pada *buffer*, bukan pada berkas.

Sebagai contoh, kita buka sebuah berkas ke dalam Vim:

```
vim welcome.js
```

Kemudian kita lakukan penyuntingan di dalamnya, entah itu menghapus, atau menambah teks ke dalamnya. Proses penyuntingan tersebut hanya akan memengaruhi *buffer*, tidak pada berkas aslinya.

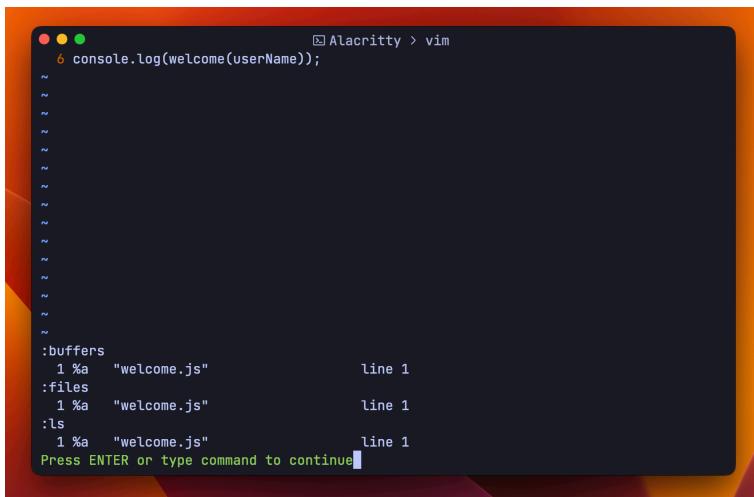
Pada editor teks semacam VS Code, ketika kita menyunting sebuah berkas, tidak akan terjadi apa-apa pada berkas aslinya sampai kita simpan perubahan tersebut, *'kan?*

Beginu juga dengan Vim, perubahan yang kita buat pada berkas yang sedang dibuka, hanya akan membuat perubahan pada *buffer*, sampai kita simpan perubahan tersebut dengan perintah :w, misalnya. Saat perubahan disimpan, berkas aslinya akan digantikan oleh isi *buffer*.

Kita dapat melihat daftar *buffer* yang ada di Vim dengan perintah :ls, :files, atau :buffers.

---

<sup>13</sup> "Koleksi Plugin Vim" <https://vimawesome.com>

A screenshot of a terminal window titled "Alacritty > vim". The window shows a list of buffers and files. The buffer list includes "welcome.js" at line 1, "welcome.js" at line 1, and "welcome.js" at line 1. The file list also includes "welcome.js" at line 1. The prompt "Press ENTER or type command to continue" is visible at the bottom.

Daftar buffer saat ini

Ketiga perintah tersebut akan melakukan hal yang sama. Setiap *buffer* dapat diidentifikasi dengan nama dan nomor.

Nama *buffer* adalah nama berkas yang terkait dengan *buffer* tersebut. Sedangkan nomor *buffer* adalah nomor urut dan unik yang diberikan oleh Vim. Nomor tersebut tidak akan berubah selama satu sesi Vim.

Sejauh ini saya memiliki dua buah berkas: *welcome.js* dan *add.js* dengan masing-masing isi seperti ini:

### *welcome.js*

```
const userName = "Nauval";

function welcome(name) {
    return `Welcome, ${name}`;
}

console.log(welcome(userName));
```

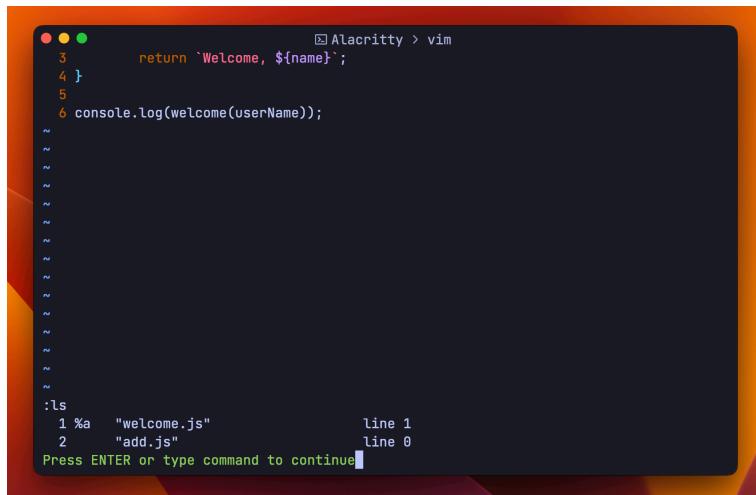
### *add.js*

```
function add(a, b) {  
    return a + b;  
}  
  
console.log(add(1, 2));
```

Untuk membuka kedua berkas tersebut, kita dapat menggunakan perintah yang sama seperti sebelumnya, hanya saja kita tambahkan nama berkas yang lain sebagai argumen kedua.

```
vim welcome.js add.js
```

Dengan seperti ini, Vim akan membuka kedua berkas dan membacanya ke dalam 2 *buffer* yang berbeda. Gunakan perintah :ls untuk mengetahuinya.

A screenshot of a terminal window titled "Alacritty > vim". The window shows two buffers. The first buffer contains code for a "welcome.js" file, starting with "function welcome(name) {". The second buffer contains code for an "add.js" file, starting with "function add(a, b) {". Below the buffers, the command ":ls" is run, followed by the output "1 %a "welcome.js"" and "2 "add.js"". The prompt "Press ENTER or type command to continue" is visible at the bottom.

Daftar buffer saat ini

Terdapat nomor dan juga tanda %a. Nomor dapat kita gunakan sebagai referensi untuk berpindah *buffer*. Sedangkan tanda %a sederhananya adalah indikator untuk memberitahu kita bahwa *buffer* tersebut yang sedang dilihat.

Dengan adanya *buffer*, Vim memungkinkan kita untuk melakukan penyuntingan beberapa berkas sekaligus dalam satu *instance* Vim

yang sama. Kita tidak perlu keluar-masuk Vim untuk menyunting satu per satu berkas.

Kita bisa beralih ke *buffer* yang lain dengan beberapa perintah sebagai berikut:

- :bn untuk *buffer next*
- :bp untuk *buffer previous*
- :b *namaberkas* untuk membuka *buffer* berdasarkan nama berkas (contoh: :buffer welcome.js)
- :bN untuk membuka *buffer* berdasarkan nomor (contoh: :buffer2)

Tentu kamu sedikit repot untuk berpindah *buffer* dengan mengetik perintah-perintah tersebut, kendati perintahnya cukup ringkas. Tapi, akan lebih cepat bila dilakukan dengan menekan tombol kombinasi.

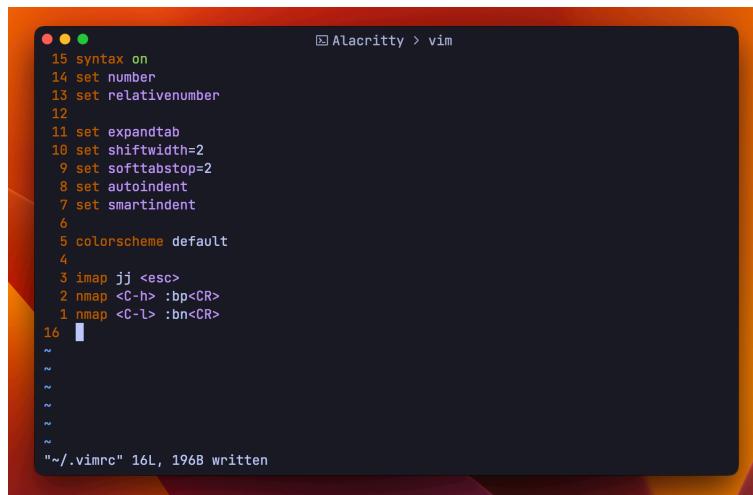
Saya me-*remap* tombol CTRL-h untuk :bp, dan tombol CTRL-l untuk :bn.

```
nmap <C-h> :bp<CR>
nmap <C-l> :bn<CR>
```

Perintah nmap akan membuat pemetaan pada mode normal; <C-h> dan <C-l> maksudnya tombol CTRL-h dan CTRL-l; :bp<CR> dan :bn<CR> maksudnya jalankan perintah bp dan bn.

Kamu dapat menambahkan konfigurasi tersebut ke berkas *vimrc*. Setiap kali melakukan perubahan pada berkas *vimrc*, kamu perlu memuat ulang konfigurasi tersebut ke Vim.

Seperti sebelumnya, setidaknya ada dua cara untuk melakukannya, kamu bisa menutup dan membuka kembali program Vim; atau menggunakan perintah :source tanpa harus menutup program Vim.

A screenshot of the Alacritty terminal window titled "Alacritty > vim". The terminal displays the contents of a vim configuration file, vimrc. The code includes syntax highlighting for Vimscript commands. The terminal window has a dark background with orange and red highlights around the title bar and scroll bar. The status bar at the bottom shows the path "~/.vimrc" and the file statistics "16L, 196B written".

```
15 syntax on
14 set number
13 set relativenumber
12
11 set expandtab
10 set shiftwidth=2
 9 set softtabstop=2
 8 set autoindent
 7 set smartindent
 6
 5 colorscheme default
 4
 3 imap jj <esc>
 2 nmap <C-h> :bp<CR>
 1 nmap <C-l> :bn<CR>
16
```

~  
~  
~  
~  
~  
~  
~  
~

"~/.vimrc" 16L, 196B written

Konfigurasi *vimrc* saat ini

Sekarang kamu dapat beralih antar *buffer* dengan kombinasi tombol **CTRL-h** dan juga **CTRL-l**.

Saat *buffer* tidak digunakan lagi, misal kamu merasa telah selesai melakukan penyuntingan pada sebuah berkas, kamu dapat menghapus *buffer* dari daftar. Gunakan perintah **:bd** untuk menghapus *buffer* yang sedang dibuka, atau **:bdN** di mana **N** adalah nomor *buffer*.

Untuk perintah **:bd**, saya melakukan *mapping* pada kombinasi tombol **CTRL-c**. Jadi, untuk menutup *buffer*, saya hanya perlu menekan tombol **CTRL-c** pada *keyboard*.

```
nmap <C-c> :bd<CR>
```

Selain perintah untuk bernavigasi dan menghapus *buffer*, tentu saja Vim memiliki perintah untuk membuat *buffer*. Hal ini yang kita akan gunakan ketika hendak membuat berkas atau membuka berkas yang ada ke dalam Vim yang sedang dibuka.

Untuk membuat *buffer*, kita dapat menggunakan perintah **:e** nama-berkas. Sebagai contoh seperti ini:

```
:e hello.py
```

Perintah tersebut akan membuat *buffer* baru. Bila kamu memiliki sebuah berkas dengan nama *hello.py* di dalam folder yang sama dengan berkas sebelumnya, maka Vim akan membuka dan membaca isi berkas tersebut ke dalam *buffer* baru. Bila Vim tidak menemukannya, maka Vim akan membuat *buffer* kosong.

Setelah menjalankan perintah tersebut, Vim akan menunculkan pesan:

```
"hello.py" [New]
```

Seandainya Vim menemukan berkas yang sudah ada dengan nama *hello.py*, kira-kira Vim akan memunculkan pesan seperti ini:

```
"hello.py" nL, xB
```

Di mana *nL* adalah total baris dari isi berkas, contoh *10L* untuk 10 baris. Sedangkan *xB* adalah ukuran berkas dalam *byte*, contoh *30B*.

Kamu akan melihat sebuah *buffer* baru pada daftar ketika menggunakan perintah `:ls`.

```
:ls
1 #   "welcome.js"          line 1
2     "add.js"              line 0
3 %a   "hello.py"           line 1
Press ENTER or type command to continue
```

Daftar buffer saat ini

Perlu diingat kembali, *buffer* adalah area memori Vim yang digunakan untuk menyimpan sementara isi berkas. Setiap perubahan yang kita lakukan di Vim itu akan diterapkan ke *buffer*, bukan berkas aslinya.

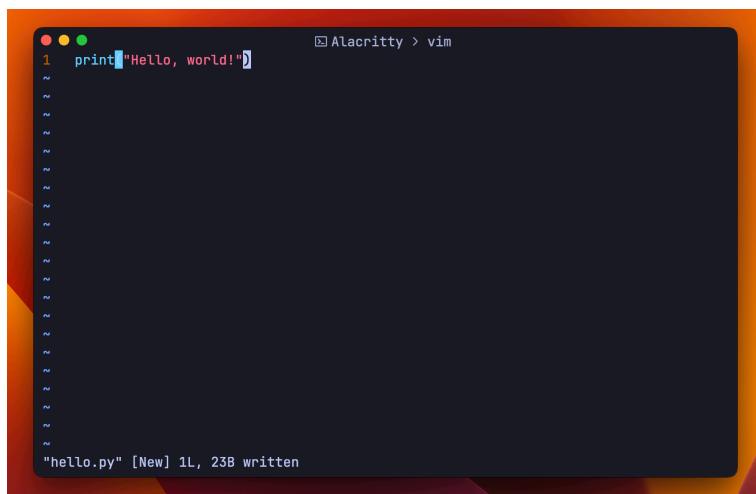
Perintah di atas bukan untuk membuat berkas, melainkan membuat *buffer* baru. Jadi, untuk menyimpan *buffer* menjadi berkas. Kita masih perlu menggunakan perintah lain, seperti :w.

Kita tidak perlu memberi nama berkas lagi pada perintah :w seperti sebelumnya, karena kita sudah memberikan nama *buffer* sebelumnya, nama *buffer* tersebut akan digunakan sebagai nama berkas saat *buffer* tersebut disimpan.

Misal kita tulis kode berikut di *buffer hello.py*:

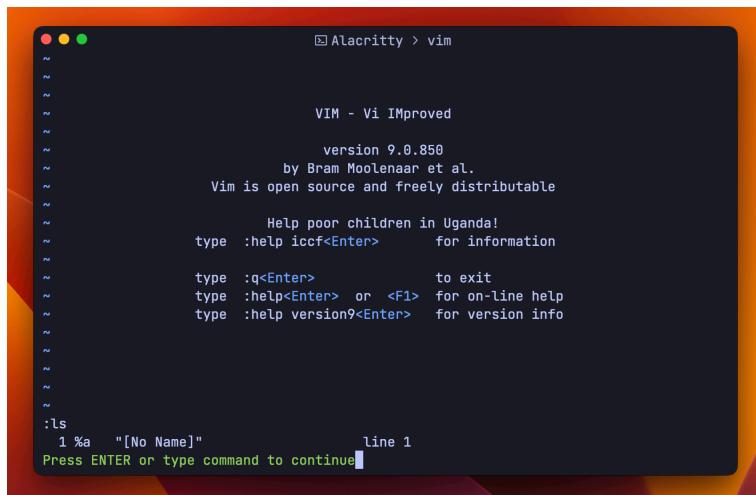
```
print("Hello, world!")
```

Simpan *buffer* tersebut dengan perintah :w.

A screenshot of a terminal window titled "Alacritty > vim". The terminal shows a single line of Python code: "print('Hello, world!')". Below the code, there are approximately 20 blank lines. At the bottom of the terminal window, the status bar displays the message "'hello.py' [New] 1L, 23B written".

Menyimpan buffer *hello.py* menjadi berkas

Kamu dapat berpikir bahwa sebuah berkas diwakili oleh *buffer* di Vim, tapi tidak semua *buffer* mewakili berkas. Faktanya, ketika kita membuka Vim pertama kali, Vim akan membuat sebuah *buffer* kosong tanpa nama.

A screenshot of a terminal window titled "Alacritty > vim". The window shows the Vim startup screen, which includes the text "VIM - Vi IMproved", "version 9.0.850", "by Bram Moolenaar et al.", "Vim is open source and freely distributable", and usage instructions for help commands. At the bottom, it shows ":ls" and "1 %a [No Name]" followed by "line 1". A message at the bottom says "Press ENTER or type command to continue".

```
VIM - Vi IMproved
version 9.0.850
by Bram Moolenaar et al.
Vim is open source and freely distributable

Help poor children in Uganda!
type :help iccf<Enter>      for information

type :q<Enter>              to exit
type :help<Enter> or <F1> for on-line help
type :help version9<Enter> for version info

:ls
1 %a [No Name]               line 1
Press ENTER or type command to continue
```

Sebuah buffer kosong saat memulai Vim

Itu artinya *buffer* tersebut tidak mewakiliki berkas apapun, sampai *buffer* tersebut kamu simpan menjadi sebuah berkas.

Mungkin sampai sini kamu berpikir bahwa *buffer* ini setara dengan fitur *tab* yang ada di editor teks semacam VS Code. Secara teknis mungkin iya. Tapi, Vim memiliki definisi lain mengenai *tab*, dan itu fitur yang sangat berbeda.

Kita akan membahas *tab* di Vim, tapi sebelum itu kita akan membahas fitur *windows* terlebih dahulu.

## Windows

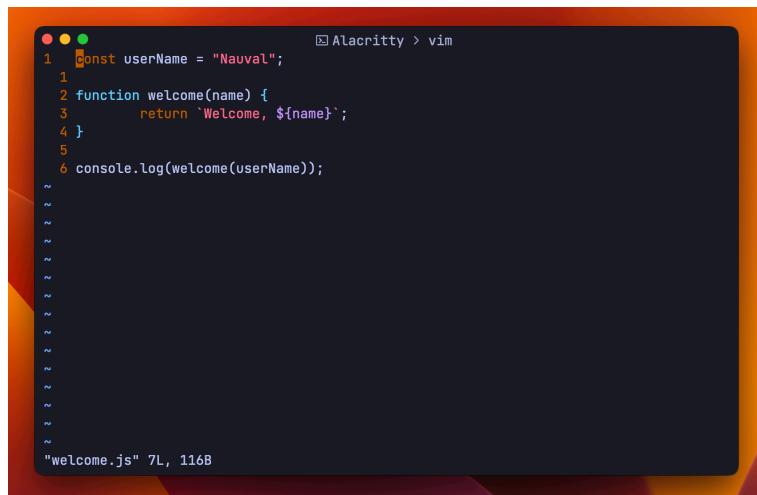
Kita sudah belajar bahwa ketika membuka berkas ke dalam Vim, maka Vim akan membaca isinya dan menahannya ke dalam *buffer*. Kemudian kita juga dapat melihat isi dari berkas tersebut di Vim. Hal ini bisa terjadi karena adanya *window*.

Di Vim, *window* adalah *viewport* atau area pandang pada *buffer*. Jadi, saat kita membuka berkas pada Vim, Vim akan memuat isi berkas tersebut ke dalam *buffer*, dan isi *buffer* tersebut akan ditampilkan kepada kita melalui *window*.

Saat kita membuka Vim seperti ini:

```
vim welcome.js
```

Vim akan membaca isi dari berkas *welcome.js* dan menaruhnya ke dalam *buffer*. Sampai sini hanya Vim yang tahu isi dari *buffer* tersebut. Kemudian Vim akan membuat sebuah *window* untuk menampilkan isi dari *buffer* tersebut kepada kita.

A screenshot of a terminal window titled "Alacritty > vim". The window displays the code of a JavaScript file named "welcome.js". The code contains a single function that logs a welcome message to the console. The terminal shows the first few lines of the file, followed by a series of tilde characters (~) indicating more content, and then the file statistics at the bottom: "welcome.js" 7L, 116B.

```
const userName = "Nauval";
function welcome(name) {
    return `Welcome, ${name}`;
}
console.log(welcome(userName));
```

Membuka berkas *welcome.js* di Vim

Kamu dapat melihat tangkapan layar di atas. Apa yang kamu lihat sekarang adalah sebuah *buffer* dan sebuah *window* untuk menampilkan isi *buffer* dari berkas *welcome.js*.

*Window adalah sebuah area pandang untuk melihat isi buffer*

Faktanya, ketika kita membuka Vim, selain ia akan membuat sebuah *buffer* kosong tanpa nama, ia juga akan membuat sebuah *window* baru.

Di Vim, *buffer* dan *window* adalah hal yang berbeda dan mereka independen. Kamu dapat membuat beberapa *window* di Vim untuk menampilkan satu *buffer* yang sama. Atau kamu juga dapat membuat beberapa *window* di Vim untuk menampilkan beberapa *buffer* yang berbeda.

Sekarang kita buka 2 berkas yang berbeda pada satu sesi Vim yang sama:

```
vim welcome.js add.js
```

Sekarang kita memiliki 2 *buffer* dan satu *window*. Sekarang kita akan membuat *window* lainnya dengan perintah `:split`.

A screenshot of the Alacritty terminal window. It shows two horizontal panes. The top pane displays the code for 'welcome.js':

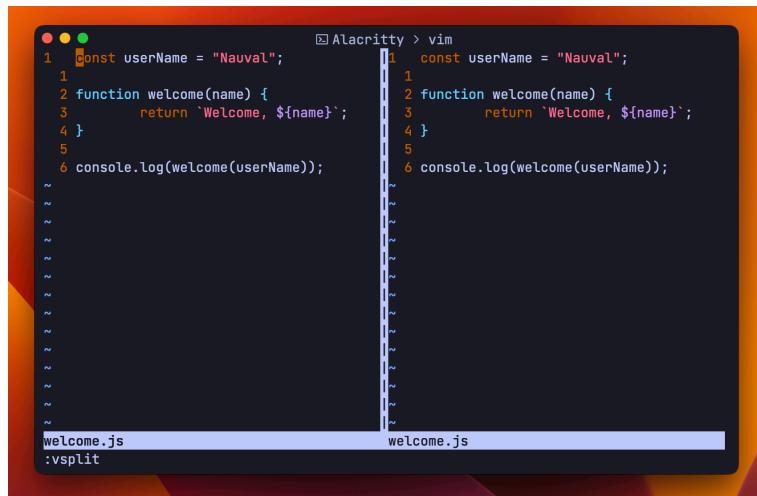
```
const userName = "Nauval";
function welcome(name) {
    return `Welcome, ${name}`;
}
console.log(welcome(userName));
```

The bottom pane also displays the same code for 'welcome.js'. In the bottom-left corner of the terminal, the command ':split' is visible, indicating the creation of the second window.

2 window menampilkan buffer yang sama

Kamu dapat melihat sekarang bahwa 2 *window* yang berbeda menampilkan *buffer* yang sama. Perintah `:split` akan membagi *window* saat ini menjadi dua secara horizontal. Saat ini perintah `:q` tidak akan menutup Vim, melainkan akan menutup *window* yang sedang aktif.

Sebagai alternatif, kamu dapat menggunakan perintah `:vsplit` untuk membagi *window* saat ini menjadi dua secara vertikal.



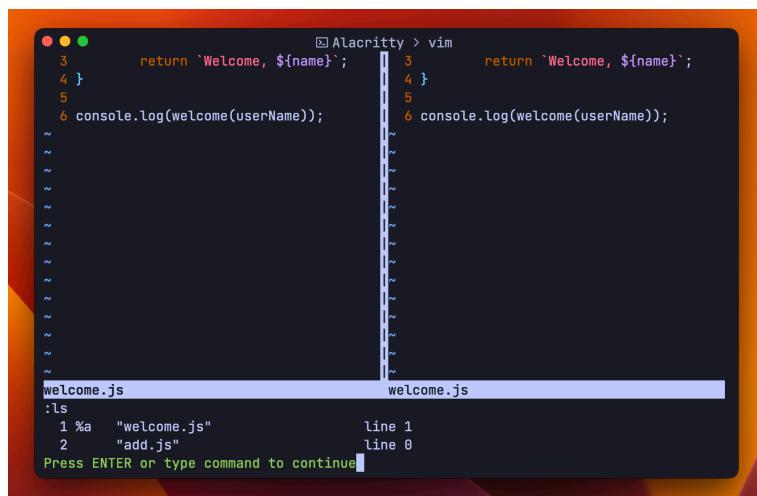
```
const userName = "Nauval";
function welcome(name) {
    return `Welcome, ${name}`;
}
console.log(welcome(userName));
```

welcome.js :vsplit

```
const userName = "Nauval";
function welcome(name) {
    return `Welcome, ${name}`;
}
console.log(welcome(userName));
```

Membagi window secara vertikal

Walaupun kamu melihat dua *window* menampilkan isi *buffer* yang sama, bukan berarti *buffer* untuk berkas *welcome.js* menjadi ganda. Kedua *window* tersebut tetap merujuk pada *buffer* yang sama.



```
return `Welcome, ${name}`;
}
5
6 console.log(welcome(userName));
```

welcome.js

```
ls
1 %a "welcome.js"           line 1
2 "add.js"                  line 0
```

Press ENTER or type command to continue

Daftar buffer saat ini

Seperti yang saya bilang sebelumnya, *buffer* dan *window* itu berbeda.

Perintah `:split` maupun `:vsplit` dapat kita berikan argumen nama berkas, misal seperti ini:

```
:split hello.py  
:vsplit add.js
```

Bila belum terdapat *buffer* yang mewakili nama berkas tersebut, maka Vim akan membaca berkas tersebut ke dalam *buffer* baru. Selain itu, Vim akan membaca *buffer* yang ada yang mewakili berkas tersebut.

Sebagai alternatif, kamu juga dapat menggunakan perintah berikut:

- `:new` untuk membuat sebuah *window* baru dengan sebuah *buffer* kosong
- `:vnew` sama seperti `:new`, namun secara vertikal

Untuk navigasi *window*, kita dapat menggunakan beberapa perintah berikut:

- `CTRL-w j` untuk memindahkan kursor ke bawah dari *window* saat ini
- `CTRL-w k` untuk memindahkan kursor ke atas dari *window* saat ini
- `CTRL-w h` untuk memindahkan kursor ke kiri dari *window* saat ini
- `CTRL-w l` untuk memindahkan kursor ke kanan dari *window* saat ini
- `CTRL-w CTRL-w` untuk memindahkan kursor ke bawah atau ke kanan dari *window* saat ini

Sejauh ini kita sudah belajar prihal *buffer* dan *window*. Saya harap kamu sudah paham mengenai kedua hal ini, seandainya belum paham juga tidak apa-apa, tidak perlu terburu-buru. Ambil waktumu.

# Tab

Memahami *tab* akan sangat mudah seandainya kamu sudah paham *buffer* dan *window*. Konsep *tab* di Vim berbeda dengan konsep *tab* yang ada di editor teks konvensional.

Editor teks seperti VS Code memiliki konsep *tab* umumnya berarti untuk membuka sebuah berkas. Jadi bisa dibilang satu *tab* itu mewakili satu *berkas*, ketika kita tutup *tab* tersebut, maka berkasnya juga akan pergi.

Sedangkan di Vim, *tab* tidak ada kaitannya sama sekali dengan *berkas*. Sebelumnya kita sudah belajar bahwa berkas diwakili oleh *buffer*. Di Vim, *tab* bersifat layaknya halaman atau tata letak saja.

Sebelumnya kita sudah membuat beberapa *window*, bukan? Mari kita lakukan lagi.

```
vim welcome.js add.js
```

Lalu bagi menjadi beberapa *window*.

```
:vsplit add.js
```

Sekarang seharusnya kamu memiliki dua *window* yang dibagi secara vertikal.

The screenshot shows a terminal window titled "Alacrity > vim". It contains two vim panes. The left pane, titled "add.js", contains the following code:

```
function add(a, b) {  
    return a + b;  
}  
console.log(add(1, 2));
```

The right pane, titled "welcome.js", contains the following code:

```
const userName = "Nauval";  
function welcome(name) {  
    return `Welcome, ${name}`;  
}  
console.log(welcome(userName));
```

The status bar at the bottom shows "add.js" and "welcome.js" with their respective file sizes: "add.js" is 5L, 64B and "welcome.js" is 5L, 64B.

Membuka buffer add.js ke dalam window berbeda

Sampai sini tidak ada yang baru, kita sudah pelajar itu semua pada bagian sebelumnya.

Sekarang kita akan buat *tab* baru dengan perintah :`tabnew`.



A screenshot of the Alacritty terminal window. The title bar reads "Alacritty > vim". The current tab is "add.js [No Name]". The terminal is in vim mode, with the command line at the bottom showing ":tabnew". The main area of the terminal is filled with the character "~".

## Sebuah tab baru

Dengan perintah tersebut kita sudah memiliki sebuah *tab* baru dan satu *buffer* baru kosong. Bila kamu perhatikan, kamu akan melihat

daftar *tab* yang kamu miliki di atas layar Vim. Saat ini, seharusnya kamu sudah memiliki dua *tab* sama seperti saya.

Di *tab* ini, kita juga dapat membuat beberapa *window* seperti sebelumnya. Misal, sebelumnya kita menggunakan :*vsplit*, sekarang kita dapat menggunakan :*split welcome.js* agar membagi *window* secara horizontal.

A screenshot of an Alacritty terminal window titled "Alacritty > vim". The terminal has two tabs open. The first tab contains the code for "welcome.js":

```
add.js 2 welcome.js
1 const userName = "Nauval";
2 function welcome(name) {
3     return `Welcome, ${name}`;
4 }
5
6 console.log(welcome(userName));
```

The second tab is titled "welcome.js" and contains a single character "I", indicating it is the current window. The status bar at the bottom shows "[No Name]".

Membagi window secara horizontal

Sekarang kita akan pergi ke *tab* sebelumnya dengan perintah :*tabp* sebagai *tab-previous*; atau juga pergi ke *tab* berikutnya dengan perintah :*tabn* sebagai *tab-next*.

Kamu bisa perhatikan, di *tab* pertama kita memiliki 2 *window* yang dibagi secara vertikal; di *tab* kedua kita memiliki 2 *window* yang dibagi secara horizontal.

Inilah yang dimaksud dengan *tab* di Vim. Vim juga menyebut *tab* sebagai *tab page*, memang sifatnya seperti halaman. Ia tidak ada kaitannya dengan berkas yang sedang dibuka, *tab* hanya tata letak saja.

Kamu juga dapat membuka sebuah berkas di *tab* baru dengan perintah :*tabnew namaberkas*, misal, :*tabnew welcome.js*. Begitu

juga dengan menutup *tab*, kamu dapat melakukannya dengan perintah `:tabclose` atau `:tabc` untuk versi yang lebih ringkas.

Ketika *tab* ditutup, *buffer* untuk berkas *welcome.js* tetap masih ada di dalam daftar *buffer*. Ini sama halnya ketika kamu menutup *window* yang sedang menampilkan *buffer*, ketika kamu menutupnya, *buffer* masih ada. Artinya ketiga hal antara *buffer*, *window* dan *tab* memang independen.

Walaupun kita memiliki beberapa *tab* yang berbeda, tetapi kita tetap saja memiliki satu daftar *buffer* yang sama. Saat kita menjalankan perintah untuk melihat daftar *buffer* saat ini di beberapa *tab* yang berbeda, maka hasilnya akan tetap sama saja.

Di dalam satu *tab* kita bisa memiliki satu atau lebih dari satu *window*, begitu juga di *tab* yang lain. Secara singkat seperti ini kira-kira:

- Berkas diwakili oleh *buffer*
- *Buffer* ditampilkan oleh *window*
- Satu atau lebih dari satu *window* dapat dibuat di dalam satu *tab*
- Vim dapat memiliki beberapa *tab* yang berisi beberapa *window*

Apakah sudah mulai pusing? Atau memang sudah pusing sejak tadi? Tidak apa-apa, ambil waktumu untuk memahami konsep ini.

## Registers

Pada bagian *operator* kita sudah membahas sedikit tentang *registers*. Kamu dapat berpikir bahwa *registers* ini seperti *clipboard*. Saat kamu menyalin teks di komputer, teks tersebut akan ditaruh ke dalam *clipboard*. Begitu juga di Vim, saat kamu menyalin dengan perintah `y`, teks akan ditaruh di *registers*.

Bisa dibilang *registers* adalah beberapa ruang memori di Vim. Bila di *clipboard* kamu hanya memiliki satu ruang, di Vim kamu dapat

menaruh teks di beberapa ruang yang berbeda dengan *registers*. Beberapa ruang tersebut masing-masing memiliki *identifier* atau nama sebagai referensi agar *registers* bisa diakses nanti.

Saat kamu menghapus teks menggunakan perintah c, d, x, atau s, teks tersebut akan dihapus dan juga akan ditaruh ke dalam *registers*. Ya, secara teknis sama seperti fitur *cut* pada komputer. Itu kenapa saat kamu menghapus teks, misal menggunakan d, lalu kamu tekan p untuk mem-paste, teks yang dihapus akan muncul.

Selain menghapus, menyalin (*yank*) dengan perintah y juga akan menaruh teks yang disalin ke dalam *registers*, itu cara Vim “mengingat”. Untuk melihat daftar *registers* saat ini, kita dapat menggunakan perintah :reg.

Kita dapat mengakses *registers* dengan perintah tanda petik " sebelum nama atau *identifier*-nya, misal "a, kita berarti mengakses *registers* bernama a. Perintah mengakses *registers* ditulis sebelum perintah yang kita inginkan. Jadi bila kita menggunakan perintah "ayy, artinya kita menyalin satu baris teks dan menaruhnya ke dalam *registers* a.

Mengakses *registers* maksudnya menaruh dan mengambil teks di dalamnya. Tidak memahami konsep *registers* di Vim tidak akan membuat kamu masuk neraka, tapi memahami konsep ini memungkinkan kamu menggunakan Vim lebih efisien lagi.

Kasus yang paling sering terjadi di kalangan pengguna Vim adalah ketika sudah menyalin teks A, lalu menghapus teks B, saat ingin mem-paste teks A, yang muncul malah teks B, karena sebelumnya menghapus teks B.

Hal seperti itu umum dan wajar terjadi, karena saat menghapus, Vim juga menaruh teks yang dihapus ke dalam *registers*. Menghapus dan menyalin di Vim akan menaruh teks ke dalam sebuah ruang *registers* bawaan yang sama. Ruang *registers* tersebut dinamakan *the unnamed register*. Jadi, saat kamu menyalin, kemudian menghapus, teks

sebelumnya yang ada di *register* ini akan digantikan dengan yang baru.

Ruang *registers* tersebut dapat diakses dengan "" (tanda petik ganda dua kali). Sebagai contoh kita dapat menggunakan perintah ""p untuk mengambil (mem-paste) isi teks di *registers* tersebut. Perintah itu juga sama saja dengan perintah p. Itu kenapa perintah p bisa dibilang semacam *shorthand* untuk perintah ""p, karena akan mengakses *registers* "".

Selain *the unnamed register*, ada juga ruang *register* lain, yaitu *10 numbered registers*. Seperti namanya, *registers* ini memiliki 10 ruang yang berbeda sesuai nomor urutnya (0–9).

Untuk nomor urut 0 atau *registers* "0 akan diisi oleh teks yang terakhir disalin (*yank*) dengan perintah y. Untuk yang lainnya *registers* "1 sampai "9 akan diisi oleh 9 teks terakhir yang dihapus. *Registers* "1 yang paling terbaru, sedangkan "9 yang paling terakhir.

Kita juga dapat membuat *registers* dengan nama sesuai alfabet, yaitu a-z. Dengan *registers* ini kita dapat mengatasi masalah yang paling umum terjadi seperti yang saya jelaskan sebelumnya. Karena menyalin dan menghapus akan menaruh ke dalam *registers* yang sama, maka kita dapat menyalin teks ke *registers* yang lain agar tidak terganti tekannya.

Misal, kita ingin menyalin satu baris dan menaruhnya ke dalam *registers* z, kita dapat menggunakan "zyy, atau hanya ingin menyalin satu kata, "ziyw untukmu. Untuk mengambil teks yang disalin tersebut, kita dapat menggunakan "zp artinya *paste* dari *registers* z.

Selain ketiga *registers* yang saya bahas di atas, Vim memiliki 7 jenis *registers* yang lain, jadi 10 jumlahnya. Tapi mempelajari ketiga *registers* adalah modal yang bagus untuk menjelajah Vim lebih jauh lagi.

# Macros

Bayangkan kamu dapat merekam beberapa urutan perintah dan mengulangnya terus-menerus dalam beberapa waktu yang berbeda. Selamat datang di *macros*, karena fitur ini memungkinkan melakukan hal tersebut.

*Macros* adalah fitur di Vim yang memungkinkan kita untuk merekam beberapa urutan perintah, lalu kita dapat mengaplikasikannya dalam waktu yang berbeda. Fitur ini sangat membantu ketika kita ingin *refactor* kode.

Setiap *macros* memiliki *identifier* yang dapat kita gunakan sebagai referensi untuk mengaplikasikan *macros* tersebut di waktu yang lain. Kita dapat mulai merekam *macros* dengan perintah `q`, dibarengi dengan *identifier*, misal `a`. Jadi, dengan perintah `qa`, kita akan merekam *macros* dengan *identifier* `a`.

Untuk mengaplikasikannya, kita dapat menggunakan perintah `@`, dibarengi dengan *identifier* yang kita atur sebelumnya, misal `a`, jadi kita dapat menggunakan perintah `@a`.

Sekarang kita akan mencobanya, pastikan kamu berada di dalam mode normal. Tekan perintah `qa`, maka kamu akan melihat indikator berikut di bawah-kiri layar Vim:

```
recording @a
```

Pada saat ini, Vim sedang menunggu perintah-perintah yang kamu masukkan. Sekarang, kita bisa masuk ke mode *insert* dengan perintah `i`, dan mulai menulis teks misal seperti ini:

```
halo nauval
```

Tekan **ENTER** untuk membuat baris baru, lalu kembali ke mode normal dengan menekan tombol `ESC` atau `jj` bila kamu mengikuti konfigurasi

yang saya tulis sebelumnya. Masukkan perintah `q` untuk membuat *macros* berhenti merekam.

Sekarang kamu berada pada mode normal dan memiliki teks seperti di atas. Untuk mengaplikasikan *macros*, kamu dapat menggunakan perintah `@a`, karena `a` adalah nama *identifier* yang kita berikan sebelumnya.

Perintah tersebut akan membuat teks yang sama, hasilnya maka akan seperti ini:

```
halo nauval  
halo nauval
```

Kamu dapat mengaplikasikan *macros* sebanyak yang kamu inginkan. Misal, kamu dapat menggunakan perintah `10@a`, untuk mengulang perintah `@a` sebanyak 10 kali.

Setiap *macros* yang direkam, Vim akan menaruhnya ke dalam *registers*. Ya, “*registers*” yang sama yang kita bahas sebelum ini. Karena kita memberikan `a` sebagai *identifier*, maka kamu seharusnya sudah dapat menebak. Ya, *macros* tersebut ditaruh di dalam *registers* `a`.

Bila kamu melihat daftar *registers* saat ini dengan perintah `:reg`, maka kamu akan melihat *registers* `a` berisikan teks kira-kira seperti ini:

```
ihalo nauval^Mjj
```

Vim menaruh perintah-perintah yang kita masukkan saat merekam *macros* sebagai teks biasa atau *plain text* saja. Tidak ada yang spesial. Itu artinya kita dapat menyalin isi *registers* tersebut untuk digunakan di mana saja, termasuk di sesi Vim yang berbeda!

Sekarang kita akan coba me-*refactor* teks sebelumnya, katakanlah kita memiliki teks seperti ini:

```
halo nauval
```

Lalu kita ingin memperbaikinya menjadi seperti ini:

Halo, Nauval!

Pertama yang kita perlu lakukan adalah posisikan kursor pada huruf “h”, lalu mulai merekam *macros* dan menaruhnya ke dalam *registers* yang lain, misal s. Maka kita dapat menggunakan perintah qs.

Saat *macros* sedang direkam, kita dapat memasukkan perintah-perintah berikut:

- ~ untuk membuat huruf “h” menjadi kapital
- e untuk memindahkan posisi kursor ke huruf “o”
- a untuk masuk ke mode *insert* dengan posisi kursor berada di depan huruf “o”
- , untuk memberikan tanda koma setelah kata “Halo”
- jj atau ESC untuk beralih ke mode normal
- w untuk memindahkan posisi kursor ke huruf “n”
- ~ untuk membuat huruf “n” menjadi kapital
- A untuk masuk ke mode *insert* dengan posisi kursor berada di akhir baris
- ! untuk memberikan tanda seru
- jj atau ESC untuk beralih ke mode normal
- q untuk menghentikan merekam *macros*

Sekarang kamu dapat membuat teks serupa lagi pada baris baru, seperti ini:

halo nauval

Kemudian pindahkan posisi kursor ke awal baris atau pada huruf “h”, lalu gunakan perintah @s untuk mengaplikasikan *macros*. Saat ini seharusnya teks sudah berubah menjadi seperti ini:

Halo, Nauval!

Bila kita melihat ke daftar *registers* saat ini, maka *registers* s akan berisi kira-kira seperti ini:

`~ea, jjw~A!jj`

Itulah sekumpulan perintah-perintah yang kita rekam sebelumnya, ketika kita gunakan perintah `@s`, maka sama saja Vim akan mengulang sekumpulan perintah-perintah itu lagi pada waktu tersebut.

*Macros* pada dasarnya mirip seperti perintah `.`, bedanya dengan perintah `.` kita hanya dapat mengulang satu perintah terakhir saja, sedangkan dengan *macros* kita dapat mengulang sekumpulan urutan perintah. Tentu kamu dapat menggunakan *macros* pada contoh kasus yang lebih kreatif lagi ketimbang ini.

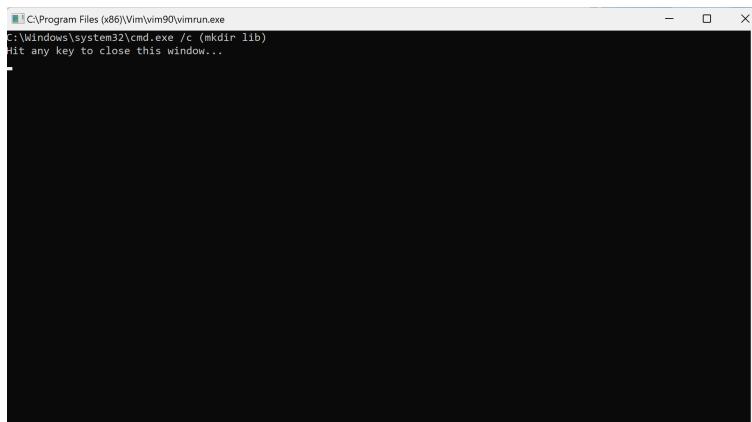
## External Command

Vim memungkinkan kamu untuk menjalankan perintah-perintah atau program CLI yang biasa kamu jalankan di dalam *shell*, seperti `mkdir`, `ls`, `git`, `curl` atau program CLI lainnya tanpa harus keluar Vim.

Kamu dapat menggunakan tanda seru atau *bang* untuk menjalankan program CLI di dalam Vim. Misal, kamu ingin membuat folder bernama `lib` dengan perintah `mkdir`, kamu dapat menggunakan perintah `:!mkdir lib`. Ketika perintah selesai dieksekusi maka akan *terminal* akan menampilkan teks:

Press ENTER or type command to continue

Di Windows akan menampilkan jendela *terminal* baru.



Jendela *terminal* baru

Tekan tombol ENTER akan membuatmu kembali ke Vim. Saat menjalankan perintah tersebut, kamu akan menjalankan perintah atau *program eksternal* di dalam Vim. Kemudian Vim akan mengalihkan tampilan ke layar *shell* untuk menampilkan STDOUT dari perintah tersebut, di Windows akan membuka jendela *terminal* baru seperti di atas.

Cara lain untuk menjalankan perintah eksternal di dalam Vim adalah menggunakan perintah :r. Pada dasarnya perintah :r adalah untuk membaca isi berkas.

Sebagai contoh kamu memiliki berkas dengan nama *welcome.js*. Dengan perintah :r *welcome.js*, Vim akan membaca isi berkas tersebut dan menulisnya ke dalam *buffer* saat ini.

Namun, apabila kamu menggunakan tanda seru atau *bang* sebagai argumen perintah :r, maka Vim akan membaca STDOUT dari perintah eksternal ke dalam *buffer* saat ini. Misal, kamu ingin mengetahui daftar berkas di dalam folder saat ini menggunakan perintah :r !ls.

Hasilnya adalah Vim akan menulis STDOUT dari perintah ls ke dalam *buffer* saat ini, kira-kira seperti ini:

```
welcome.js
hello.js
```

```
file1.txt  
file2.py
```

Tentu saja ini tergantung dari isi folder yang kamu miliki.

Contoh lain, misal kamu ingin menjalankan `curl` di dalam Vim, seperti ini:

```
:r !curl -s 'https://jsonplaceholder.typicode.com/todos/1'
```

Maka hasil dari `curl` tersebut akan ditulis ke dalam *buffer* saat ini:

```
{  
    "userId": 1,  
    "id": 1,  
    "title": "delectus aut autem",  
    "completed": false  
}
```

Kebalikan dari perintah `:r`, kita dapat menggunakan perintah `:w` untuk menjalankan perintah eksternal di dalam Vim.

Sebelumnya kita mempelajari bahwa perintah `:w` ini untuk menyimpan *buffer* saat ini menjadi berkas. Namun, apabila kita menggunakan tanda seru atau *bang* sebagai argumen perintah `:w`, maka Vim akan mengambil isi dari *buffer* saat ini dan dijadikan sebagai STDIN perintah eksternal.

Misal, *buffer* saat ini adalah kode JavaScript seperti ini:

```
const userName = "Nauval";  
  
function welcome(name) {  
    return `Welcome, ${name}`;  
}  
  
console.log(welcome(userName));
```

Dan kamu ingin menjalankan kode JavaScript tersebut di dalam *environment* Node JS dengan program node. Kamu dapat menggunakan perintah :w !node.

Vim akan memunculkan hasil seperti ini:

```
Welcome, Nauval  
Press ENTER or type command to continue
```

Baris pertama pada teks di atas adalah STDOUT dari perintah node .

Ketiga cara di atas sama saja, karena pada dasarnya untuk menjalankan perintah eksternal di dalam Vim adalah menggunakan tanda seru atau *bang*, yang membedakan adalah bagaimana kita ingin membaca STDOUT atau menulis STDIN pada perintah eksternal tersebut.

## Command-line Mode

Sejauh ini kita sudah belajar banyak hal pada Vim, mencari teks dengan perintah / atau ?, melihat daftar *buffer* dengan :ls, atau menyalakan fitur *syntax* pada Vim. Hal-hal tersebut termasuk ke dalam mode *command-line* di Vim.

Mode *command-line* juga termasuk ke dalam mode, sama seperti mode normal, mode *insert*, atau mode *visual*. Ketika kamu berada di mode ini, posisi kursor akan pindah ke bawah layar di mana kamu dapat menulis perintah-perintah Vim.

Kamu dapat masuk ke mode ini pada saat menggunakan perintah pencarian seperti ? atau /; tanda titik dua :; atau tanda seru !. Kamu dapat masuk ke mode ini hanya pada mode *visual* atau mode normal. Gunakan tombol ESC, CTRL-c, atau CTRL-[ untuk keluar dari mode *command-line*.

Vim juga memungkinkan kamu untuk mengulang perintah terakhir yang dieksekusi di mode *command-line*. Misal, terakhir kamu menjalankan perintah `:ls`, kamu dapat menggunakan perintah `@:` untuk menjalankan perintah itu lagi, tanpa harus mengulang mengetik `:ls` kembali.

Selain itu, kamu juga dapat mengakses *registers* dan menggunakan *autocomplete* pada mode ini. Seandainya kamu sudah menyalin teks dan menaruhnya pada *registers* `a`, lalu kamu ingin mem-paste-nya di dalam mode *command-line*, kamu dapat menggunakan perintah `CTRL-r a`.

Tombol kombinasi `CTRL-a` memungkinkan kamu untuk mengakses *registers*, sedangkan `a` adalah *identifier* dari *registers* yang ingin diakses. Artinya, kamu dapat mengakses *registers* lain, misal the *unnamed registers* atau *registers* bawaan, kamu dapat menggunakan perintah `CTRL-r "`.

Sedangkan untuk menggunakan *autocomplete* di mode *command-line*, kamu cukup tekan tombol `TAB` di *keyboard*. Untuk melakukan *autocomplete* pada perintah *colorscheme*, kita dapat mengetik perintah `col` dan tekan `TAB`. Kamu akan memiliki dua opsi, karena terdapat dua perintah dengan awalan `col:` `colder`, dan `colorscheme`.

Kamu dapat menggunakan tombol `TAB` atau tombol kombinasi `CTRL-n` untuk mencari opsi berikutnya; juga dapat menggunakan `SHIFT-TAB` atau `CTRL-p` untuk mencari opsi sebelumnya.

Terakhir, tekan tombol arah atas atau arah bawah pada mode *command-line* untuk menjelajahi daftar riwayat perintah-perintah yang sebelumnya pernah digunakan. Untuk melihat daftar yang lebih jelas, gunakan perintah `:his`.

Vim merekam 50 perintah terakhir, kamu dapat mengubah opsi ini dengan perintah `:set history=100` untuk merekam 100 perintah terakhir.

Selain perintah :his, kamu juga dapat menggunakan perintah q: untuk membuka daftar riwayat perintah yang dapat dicari dan disunting di *window* yang terpisah.

Pada *window* tersebut kamu dapat menggunakan perintah / atau ? untuk mencari perintah yang sebelumnya kamu gunakan. Gunakan :q untuk keluar dari *window* tersebut.

# Neovim

Kita sudah selesai membahas Vim dari awal hingga bagian terakhir. Pada bagian ini kita akan membahas Neovim dengan pendekatan yang pragmatis, karena sebelumnya kita sudah mempelajari Vim dengan pendekatan yang filosofis. Pada buku ini kita tidak akan membahas Neovim secara mendalam, karena saya tidak bertujuan untuk menulis ulang dokumentasi dari Neovim.

Seperti yang sudah dijelaskan sebelumnya bahwa cara mengoperasikan Neovim sama seperti Vim, ini karena Neovim dirancang untuk menjadi *fork* dari Vim dengan tambahan fitur dan peningkatan, tapi dengan antarmuka dan perintah-perintah yang familiar.

Namun perlu diingat bahwa mungkin terdapat fitur atau perintah yang berbeda di antara keduanya, karena Neovim menambah fungsionalitas yang tidak tersedia di Vim. Salah satu perbedaan signifikan dari keduanya adalah Neovim mendukung *asynchronous plugin* yang memungkinkan *plugin* berjalan di latar belakang tanpa memblokir proses editor utama.

Selain itu, Neovim juga menambahkan fitur lain, seperti *floating window*, dukungan LSP, *debugger*, dan *terminal emulator* secara bawaan.

Neovim adalah sebuah editor teks *open-source* yang ditulis dalam bahasa pemrograman C dan dapat diperluas menggunakan bahasa *scripting* Vimscript atau bahasa lain yang dapat berinteraksi dengan API editor. Arsitektur yang lebih fleksibel yang memungkinkan pengguna untuk menyesuaikan dan memperluas editor dengan lebih mudah. Neovim populer di kalangan banyak pengguna Vim yang ingin memanfaatkan fitur dan peningkatan barunya, sambil tetap menggunakan antarmuka dan perintah-perintah Vim yang familiar.

Pada dasarnya di bagian ini kita akan menjadikan Neovim sebagai IDE kita sehari-hari, tentu saja menulis konfigurasi Neovim dari awal untuk mencapai fitur-fitur seperti yang dimiliki VS Code akan sangat

melelahkan, untuk itu kita akan menggunakan *layer* IDE Neovim yang bernama AstroNvim.

Hal-hal yang sudah kita pelajari di bagian sebelumnya sudahlah cukup sebagai modal untuk menggunakan Neovim sebagai IDE sehari-hari. Karena pada dasarnya kedua editor teks ini sama, hanya saja yang satu lebih *modern* – ada alasan awalan "neo" disematkan pada Neovim.

# Memasang Neovim

Setiap sistem operasi memiliki cara yang berbeda untuk memasang Neovim. Beruntungnya, bila kita pergi ke dokumentasi pemasangan Neovim di Github, kita dapat melihat langkah-langkah untuk memasang Neovim untuk pelbagai sistem operasi yang berbeda. Di buku ini saya akan merangkumnya untuk 3 sistem operasi yang berbeda, yaitu macOS, Windows, dan Unix-*like*.

Membuat langkah-langkah yang rinci untuk sistem operasi Unix-*like* seperti Linux akan terlalu luas jangkauannya, mengingat Linux memiliki banyak sekali distribusi yang berbeda-beda. Masing-masing distro memiliki *package manager* yang berbeda, sehingga akan membuat langkah-langkahnya pun berbeda dari yang lain.

Di buku ini, saya hanya mencontohkan memasang Neovim pada distribusi Ubuntu. Untuk distribusi yang lain seharusnya langkah-langkahnya tidak berbeda jauh. Untuk itu saya menyarankan untuk pergi ke halaman dokumentasi pemasangan Neovim<sup>14</sup> resmi untuk panduan lengkapnya. Bagi pengguna WSL dapat mengikuti langkah-langkah yang sama seperti pengguna Ubuntu.

Karena kita akan menggunakan konfigurasi AstroNvim, maka kita perlu memasang Neovim dengan versi setidaknya 0.8.

## macOS

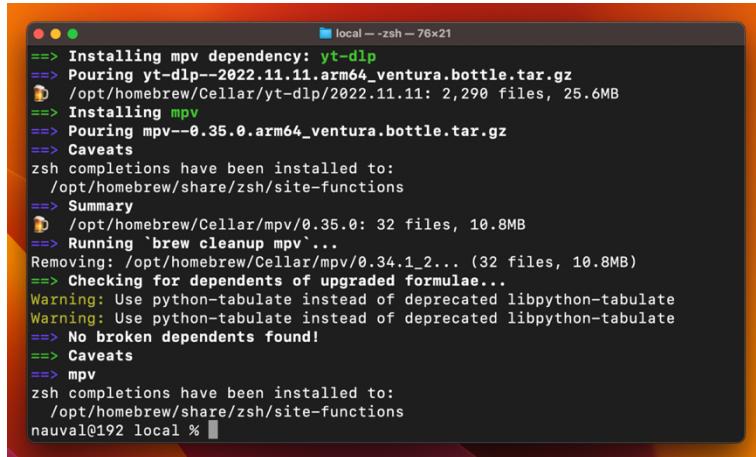
Kita dapat memasang Neovim melalui *package manager* Homebrew, pastikan *package manager* tersebut sudah terpasang. Jika belum, jangan sungkan untuk kembali membaca bagian awal buku ini tentang memasang Homebrew di macOS.

1. Buka *program* Terminal

---

<sup>14</sup> "Memasang Neovim" <https://github.com/neovim/neovim/wiki/Installing-Neovim>

2. Gunakan perintah `brew install neovim` untuk memasang Neovim melalui Homebrew, tunggu proses pemasangan hingga selesai



```
local -- zsh -- 76x21
==> Installing mpv dependency: yt-dlp
==> Pouring yt-dlp--2022.11.11.arm64_ventura.bottle.tar.gz
  /opt/homebrew/Cellar/yt-dlp/2022.11.11: 2,290 files, 25.6MB
==> Installing mpv
==> Pouring mpv--0.35.0.arm64_ventura.bottle.tar.gz
==> Caveats
zsh completions have been installed to:
  /opt/homebrew/share/zsh/site-functions
==> Summary
  /opt/homebrew/Cellar/mpv/0.35.0: 32 files, 10.8MB
==> Running `brew cleanup mpv`...
Removing: /opt/homebrew/Cellar/mpv/0.34.1_2... (32 files, 10.8MB)
==> Checking for dependents of upgraded formulae...
Warning: Use python-tabulate instead of deprecated libpython-tabulate
Warning: Use python-tabulate instead of deprecated libpython-tabulate
==> No broken dependents found!
==> Caveats
==> mpv
zsh completions have been installed to:
  /opt/homebrew/share/zsh/site-functions
nauval0192 local %
```

Proses pemasangan Neovim selesai

3. Verifikasi bahwa Neovim sudah terpasang dengan menjalankan perintah `nvim --version`. Perintah tersebut akan mencetak versi Neovim yang terpasang

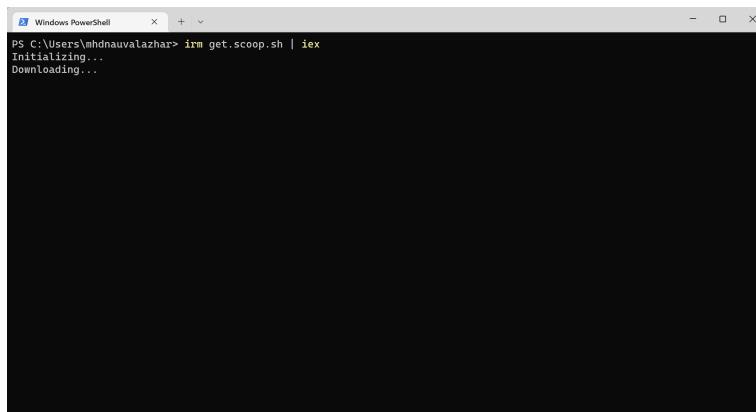
## Windows

Neovim dapat dipasang melalui *package manager* bernama Scoop di Windows. Scoop memiliki peran yang sama seperti Homebrew atau *package manager* lainnya. Selain Scoop, Windows memiliki beberapa *package manager* lainnya seperti Winget atau Chocolatey.

### Memasang Scoop

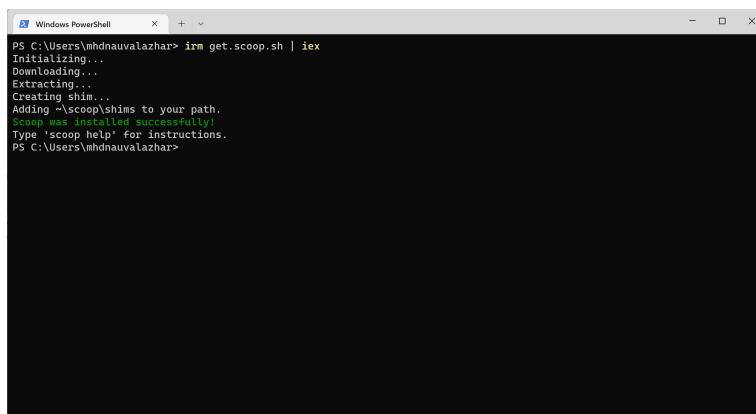
Berikut ini langkah-langkah untuk memasang Scoop di Windows melalui PowerShell.

1. Buka *program* PowerShell dan jalankan perintah `Set-ExecutionPolicy RemoteSigned -Scope CurrentUser`
2. Pasang Scoop *package manager* melalui PowerShell



Mengunduh dan memasang Scoop *package manager*

3. Tunggu hingga selesai dan jika sudah selesai akan tampil seperti ini:

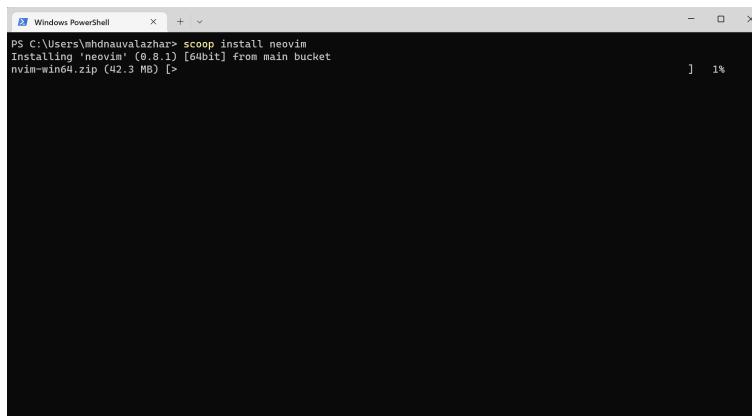


Scoop berhasil dipasang

## Memasang Neovim

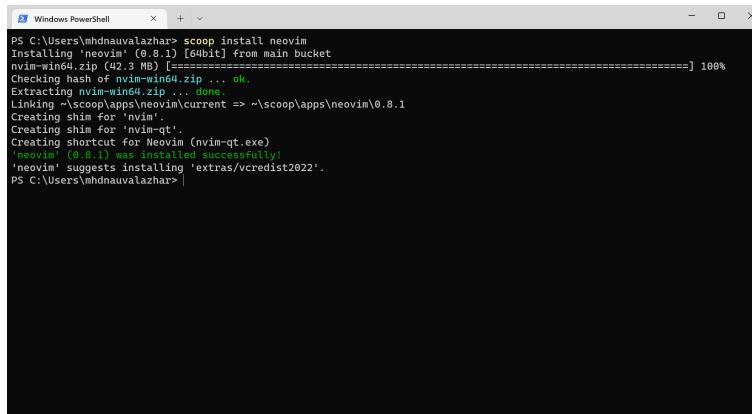
Setelah Scool terpasang, kita dapat memasang Neovim dengan langkah-langkah berikut:

1. Jalankan perintah `scoop install neovim` di dalam PowerShell



Mengunduh dan memasang Neovim

2. Tunggu hingga selesai dan jika sudah selesai akan tampil seperti ini:



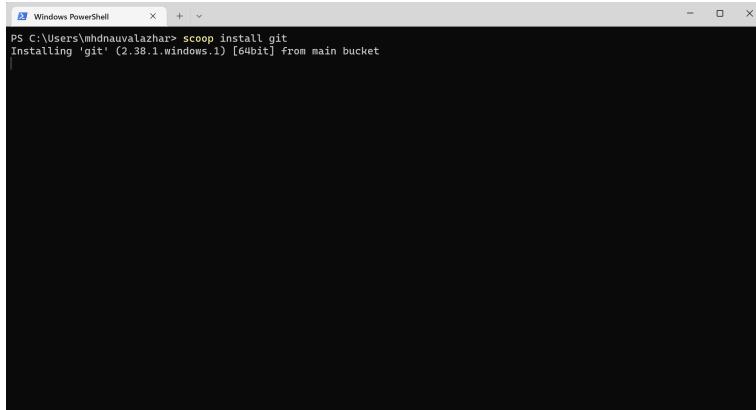
Berhasil memasang Neovim

Seperti yang tertera pada pesan di dalam PowerShell bahwa Neovim menyarankan untuk kita memasang paket *vcredist2022*. Untuk itu kita akan memasangnya, namun kita perlu memasang Git terlebih dahulu agar dapat memasang paket tersebut.

## Memasang Git

Kita memerlukan Git karena paket *vcredist2022* berada dalam *bucket* yang berbeda, Scoop membutuhkan itu.

1. Jalankan perintah `scoop install git` pada PowerShell



```
Windows PowerShell
PS C:\Users\mhdnaulazhar> scoop install git
Installing 'git' (2.38.1.windows.1) [64bit] from main bucket
```

Mengunduh dan memasang Git

2. Apabila kamu mengalami *error* mengenai *program* 7-Zip pada saat mengekstrak paket, kamu dapat memasang *program* 7-Zip secara manual dengan pergi ke situs web resminya dan unduh sesuai dengan sistem operasi yang kamu gunakan

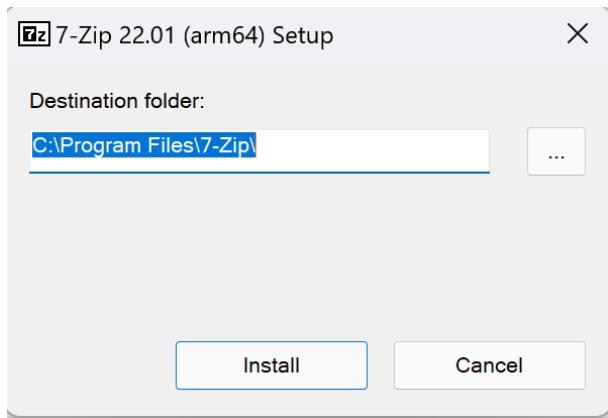
**7-Zip** is a file archiver with a high compression ratio.

**Download 7-Zip 22.01 (2022-07-15) for Windows:**

Link	Type	Windows	Size
<a href="#">Download</a>	.exe	64-bit x64	1.5 MB
<a href="#">Download</a>	.exe	32-bit x86	1.2 MB
<a href="#">Download</a>	.exe	64-bit ARM64	1.5 MB

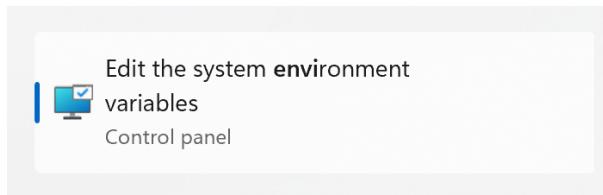
Daftar versi program 7-Zip

3. Pasang *program* tersebut seperti pada umumnya, dan catat alamat di mana *program* tersebut kamu pasang. Pada kasus saya lokasinya di `C:\Program Files\7-Zip\`. Catat baik-baik alamat tersebut karena kita akan membutuhkannya nanti.



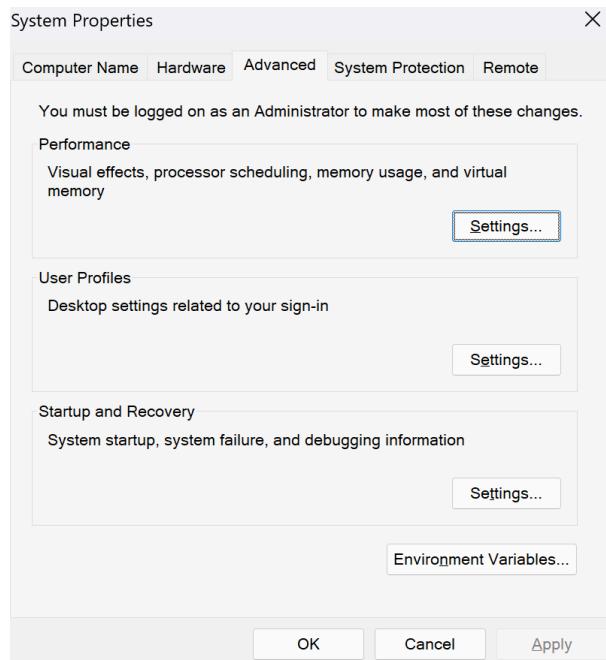
Memasang program 7-Zip

4. Jika sudah berhasil dipasang, berikutnya buka *Edit the system environment variables* melalui *start menu*



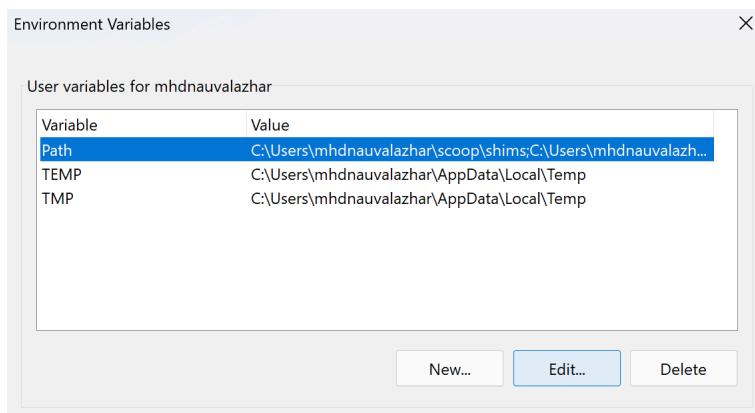
Mencari Edit the system environment variables di start menu

5. Klik tombol *Environment Variables...* yang berada di bawah



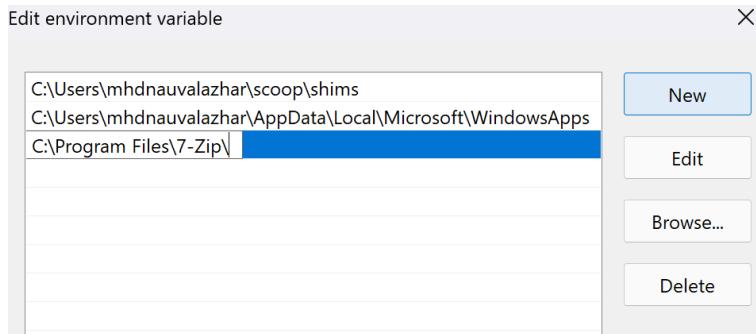
Jendela System Properties

#### 6. Pilih *Path* dan klik tombol *Edit*



Jendela Environment Variables

#### 7. Klik tombol *New* dan masukkan alamat *program 7-Zip* yang sebelumnya dipasang



Jendela Edit environment variable

8. Tutup dan buka kembali PowerShell, jalankan perintah `scoop config use_external_7zip true` untuk memberitahu Scoop agar menggunakan *program* 7-Zip yang kita pasang secara mandiri
9. Pasang kembali Git dengan perintah sebelumnya: `scoop install git`
10. Tunggu hingga proses ekstraksi selesai, bila sudah selesai akan seperti ini:

```
PS C:\Users\mhdnaualzhar> scoop config use_external_7zip true
use_external_7zip has been set to "true"
PS C:\Users\mhdnaualzhar> scoop install git
WARN: Pending previous failed installation of git.
scoop 'git' isn't installed correctly
Removing older version (2.38.1.windows.1).
'git' was uninstalled.
Installing 'git' (2.38.1.windows.1) [64bit] from main bucket
Loading PortableGit-2.38.1-64-bit.7z.exe from cache
Checking hash of PortableGit-2.38.1-64-bit.7z.exe ... ok.
Extracting d1.7z ... done.
Linking ~\scoop\apps\git\current => ~\scoop\apps\git\2.38.1.windows.1
Creating shim for 'git'.
Creating shim for 'git'.
Creating shim for 'git'.
Creating shim for 'git'.
Creating shim for 'git-gui'.
Creating shim for 'scalar'.
Creating shim for 'tig'.
Creating shim for 'git-bash'.
Creating shortcut for Git Bash (git-bash.exe)
Creating shortcut for Git GUI (git-gui.exe)
'git' (2.38.1.windows.1) was installed successfully!
Notes
Set Git Credential Manager Core by running: "git config --global credential.helper manager-core"
PS C:\Users\mhdnaualzhar>
```

Berhasil memasang Git

## Memasang VC Redist

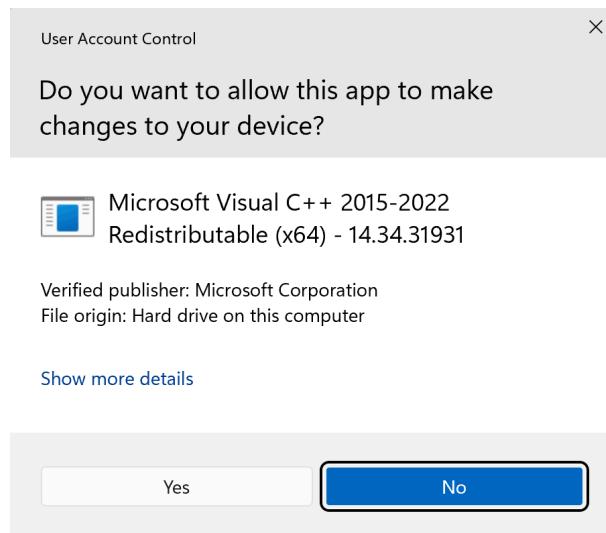
Setelah Git terpasang, kita dapat menambah *bucket extras* untuk mendapatkan paket seperti *vcredist2022*. Berikut langkah-langkahnya:

1. Buka kembali PowerShell
2. Jalankan perintah berikut `scoop bucket add extras` dan tunggu hingga selesai

```
Windows PowerShell
PS C:\Users\mhdnaualzahar> scoop bucket add extras
Checking repo... OK
The extras bucket was added successfully.
PS C:\Users\mhdnaualzahar>
```

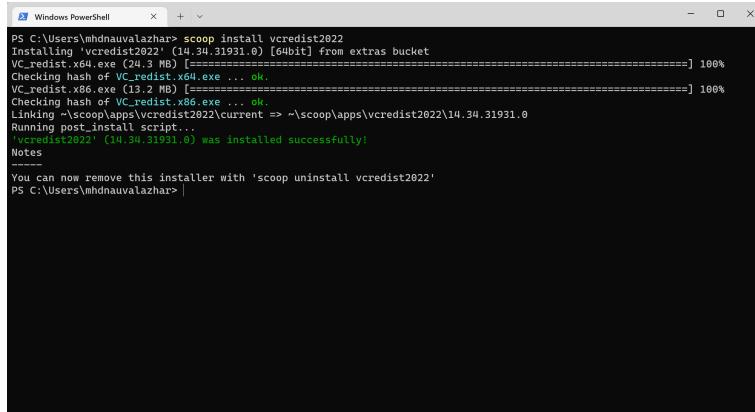
Menambahkan bucket extras

3. Jalankan perintah `scoop install vcredist2022` untuk mengunduh dan memasang paket `vcredist2022`
4. Tunggu proses pemasangan hingga selesai, klik tombol *Yes* apabila muncul *dialog User Account Control*



Dialog User Account Control

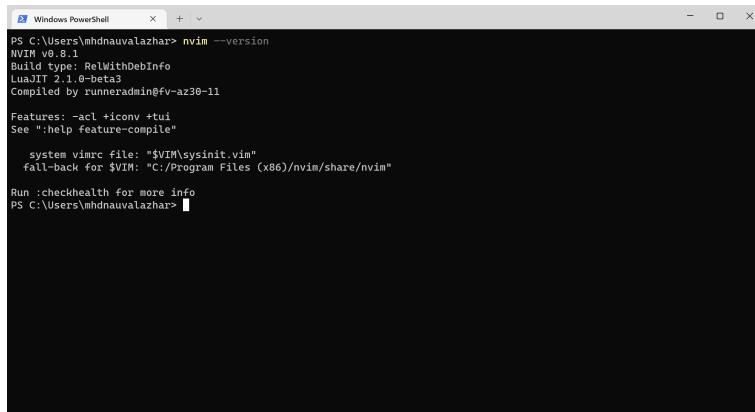
5. Bila muncul yang lainnya, lakukan hal yang sama hingga proses selesai seperti ini:



```
PS C:\Users\mhdnaualzahar> scoop install vcredist2022
Installing 'vcredist2022' (14.34.31931.0) [64bit] from extras bucket
VC_redist.x64.exe ... ok [=====] 100%
Checking hash of VC_redist.x64.exe ... ok [=====] 100%
VC_redist.x86.exe (13.2 MB) [=====] 100%
Checking hash of VC_redist.x86.exe ... ok [=====]
Linking ~\scoop\apps\vcredist2022\current => ~\scoop\apps\vcredist2022\14.34.31931.0
Running post_install script...
'vcredist2022' (14.34.31931.0) was installed successfully!
Notes
-----
You can now remove this installer with 'scoop uninstall vcredist2022'
PS C:\Users\mhdnaualzahar>
```

Berhasil memasang paket *vcredist2022*

Lakukan verifikasi pemasangan Neovim dengan perintah `nvim --version`, seharusnya akan menampilkan versi Neovim yang terpasang.



```
PS C:\Users\mhdnaualzahar> nvim --version
NVIM v0.8.1
Build type: RelWithDebInfo
LuaJIT 2.1.0-beta3
Compiled by runneradmin@fv-az30-11

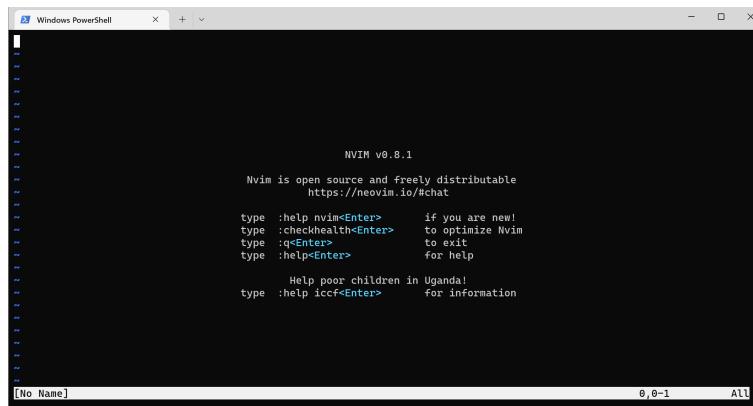
Features: -acl +iconv +tui
See ":help feature-compile"

  system vimrc file: "$VIM\sysinit.vim"
  fall-back for $VIM: "C:/Program Files (x86)/nvim/share/nvim"

Run :checkhealth for more info
PS C:\Users\mhdnaualzahar>
```

Memverifikasi pemasangan Neovim

Untuk membuka *program* Neovim, kita dapat menggunakan perintah `nvim`.



```
NVIM v0.8.1
Nvim is open source and freely distributable
  https://neovim.io/#chat

  type :help nvim<Enter>      if you are new!
  type :checkHealth<Enter>    to optimize Nvim
  type :q<Enter>            to exit
  type :help<Enter>          for help

  Help poor children in Uganda!
  type :help iccf<Enter>     for information
```

Membuka program Neovim

Sejauh ini kita sudah berhasil memasang Neovim di Windows.

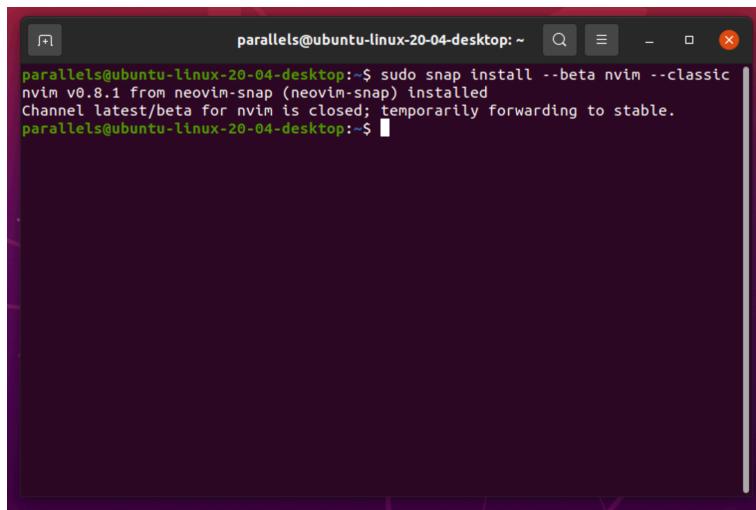
## Unix-like

Seperti yang sudah dijelaskan sebelumnya, saya hanya membuat langkah-langkah untuk Linux Ubuntu saja. Di Ubuntu, kita dapat memasang Neovim melalui APT atau Snapcraft, di sini saya memilih menggunakan Snapcraft karena memiliki versi Neovim yang kita butuhkan, yaitu versi 0.8.

Berikut langkah-langkah memasang Neovim di Ubuntu:

### Snapcraft

1. Buka *program* Terminal
2. Jalankan perintah `sudo snap install --beta nvim --classic`



```
parallels@ubuntu-linux-20-04-desktop: ~
parallels@ubuntu-linux-20-04-desktop: $ sudo snap install --beta nvim --classic
nvim v0.8.1 from neovim-snap (neovim-snap) installed
Channel latest/beta for nvim is closed; temporarily forwarding to stable.
parallels@ubuntu-linux-20-04-desktop: $
```

Berhasil memasang Neovim melalui Snapcraft

3. Verifikasi bahwa Neovim sudah terpasang dengan menjalankan perintah `nvim --version`. Perintah tersebut akan mencetak versi Neovim yang terpasang

## APT

1. Pastikan *program* Terminal terbuka
2. Jalankan perintah `sudo apt install software-properties-common`
3. Untuk menambahkan PPA Neovim, jalankan perintah `sudo add-apt-repository ppa:neovim-ppa/stable`
4. Jalankan perintah `sudo apt update` untuk memperbarui informasi paket
5. Sekarang kita dapat memasang Neovim melalui APT dengan perintah `sudo apt install neovim`
6. Verifikasi bahwa Neovim sudah terpasang dengan menjalankan perintah `nvim --version`. Perintah tersebut akan mencetak versi Neovim yang terpasang

# Konfigurasi Neovim

Sama seperti Vim, Neovim dapat dikonfigurasikan sesuai dengan preferensi dan kebutuhan kita sendiri. Selain Vimscript, Neovim memungkinkan kita menulis konfigurasi dengan bahasa Lua, berkas konfigurasi tersebut disimpan dalam folder `~/.config/nvim/init.lua` di sistem operasi Unix atau Unix-like dan `~/AppData/Local/nvim/init.lua` di Windows. Namun, kita juga dapat menentukan lokasi berkas konfigurasi yang berbeda dengan mengatur variabel `$XDG_CONFIG_HOME` atau `$NVIM_CONFIG_DIR`.

Sebelumnya kita sudah belajar menulis konfigurasi Vim dengan Vimscript, anggap saja kita memiliki konfigurasi Vim sebelumnya seperti ini:

```
set nu  
set shiftwidth = 2
```

Jika kita simpan berkas tersebut di dalam lokasi, misal, `~/.config/nvim/init.vim` maka konfigurasi tersebut tetap akan terbaca dengan baik oleh Neovim. Konfigurasi yang sama dapat kita tulis dengan bahasa Lua, menjadi seperti ini:

```
vim.opt.number = true;  
vim.opt.shiftwidth = 2;
```

Ketimbang menggunakan ekstensi `.vim` kita dapat menggunakan ekstensi `.lua`, karena kita menggunakan bahasa Lua untuk menulis konfigurasi tersebut.

Pada kasus nyata, umumnya kita akan menulis banyak opsi di dalam konfigurasi Neovim, untuk itu kita dapat mengorganisasikan penulisan opsi-opsi tersebut menggunakan *table* di dalam bahasa pemrograman Lua, contohnya seperti ini:

```
local options = {
```

```
    title = true,  
    backup = false,  
    cmdheight = 1,  
    fileencoding = "utf-8",  
    hlsearch = true,  
    ignorecase = true,  
}
```

Pada kode di atas, kita membuat sebuah variabel lokal bernama `options` yang menampung data opsi-opsi yang hendak kita gunakan. Sampai sini, kita belum memberitahu Neovim untuk mengatur opsi-opsi tersebut.

Berikutnya, kita dapat membuat perulangan dengan `for` untuk menentukan opsi Neovim secara dinamis berdasarkan data pada *table* sebelumnya, seperti ini:

```
for k, v in pairs(options) do  
    vim.opt[k] = v  
end
```

Dengan seperti contoh di atas, penulisan opsi pada Neovim akan lebih terorganisasi dengan baik. Pada dasarnya, untuk mengatur opsi pada konfigurasi Neovim dengan bahasa Lua, kita dapat menggunakan sintaksis `vim.opt.nama_opsi = nilai`.

Tentu saja untuk menulis konfigurasi agar Neovim kita dapat dijadicikan IDE akan membutuhkan waktu yang lama. Hal yang bermanfaat apabila kita ingin mempelajari Neovim secara mendalam dan menulis semua konfigurasinya dari awal secara mandiri, namun prioritas buku ini untuk mengajak kamu menggunakan Neovim sebagai IDE, bukan menjadi seorang ahli Neovim. Sekali lagi, hidup terlalu pendek untuk menulis konfigurasi Neovim dari awal.

Untuk itu kita akan menggunakan AstroNvim untuk membantu kita menjadikan Neovim sebagai IDE!

# AstroNvim

AstroNvim adalah sebuah *layer IDE* untuk Neovim. Ia menyediakan lingkungan pengembangan yang lengkap dan mudah digunakan untuk Neovim tanpa perlu menulis konfigurasi dari awal. Ia juga menyediakan berbagai fitur yang berguna untuk pengembangan, seperti *syntax highlighting*, *code completion*, dan masih banyak lagi.

Selain itu, AstroNvim memiliki fitur navigasi yang memudahkan kita untuk menavigasi melalui proyek, seperti pencarian berkas dan fungsi, dan pergi ke definisi. Dengan AstroNvim, kita dapat memulai mengembangkan proyek kita dengan cepat dan mudah di Neovim. Ini menghemat waktu dan usaha yang dibutuhkan untuk mengkonfigurasi Neovim sebagai IDE.

AstroNvim menyediakan berbagai fitur untuk membantu kita dalam pengembangan proyek menggunakan Neovim. Beberapa fitur yang disediakan oleh AstroNvim adalah:

- *File explorer* dengan Neo-tree: Ini adalah fitur *file explorer* yang menggunakan *plugin* Neo-tree untuk memudahkan kita menavigasi melalui proyek kita. Ini memungkinkan kita untuk melihat struktur direktori, membuka berkas, dan melakukan aksi lainnya dengan mudah.
- *Autocompletion* dengan CMP: Ini adalah fitur *autocompletion* yang menggunakan *plugin* CMP untuk memberikan saran kode saat kita mengetik. Ini membantu kita menulis kode dengan lebih cepat dan tepat.
- Integrasi Git dengan Gitsigns: Ini adalah fitur integrasi dengan Git yang menggunakan *plugin* Gitsigns untuk menampilkan status dari berkas di proyek kita. Ini membantu kita memantau perubahan pada berkas dan melakukan aksi Git seperti meng-*commit* atau menge-*stash* dengan mudah.
- *Statusline* dengan Heirline: Ini adalah fitur *statusline* yang menggunakan *plugin* Heirline untuk menampilkan informasi

tentang *buffer* yang sedang aktif, seperti nama berkas, mode editor, dan lainnya. Ini membantu kita memantau *buffer* yang sedang aktif dan melakukan aksi dengan lebih cepat.

- Terminal dengan Toggleterm: Ini adalah fitur terminal yang menggunakan *plugin* Toggleterm untuk memungkinkan kita menjalankan perintah shell di dalam Neovim. Ini membantu kita mengeksekusi perintah tanpa perlu keluar dari Neovim.
- *Fuzzy finding* dengan Telescope: Ini adalah fitur *fuzzy finding* yang menggunakan *plugin* Telescope untuk memudahkan kita menemukan berkas, fungsi, dan elemen lain di dalam proyek. Ini membantu kita menavigasi proyek dengan lebih cepat dan mudah.
- *Syntax highlighting* dengan Treesitter: Ini adalah fitur syntax highlighting yang menggunakan *plugin* Treesitter untuk menampilkan kode dengan warna yang berbeda sesuai dengan tipe elemennya.

AstroNvim bukan satu-satunya layer IDE yang tersedia untuk Neovim. Ada beberapa *layer* IDE lain yang tersedia untuk Neovim, seperti LunarVim, NvChad, dan masih banyak lagi. Masing-masing *layer* IDE ini menyediakan fitur yang berbeda-beda untuk membantu kita dalam pengembangan proyek di Neovim.

Jadi, kita dapat memilih *layer* IDE yang sesuai dengan kebutuhan dan preferensi kita. AstroNvim merupakan salah satu pilihan yang baik untuk *layer* IDE, tetapi bukan satu-satunya pilihan yang ada.

Awalnya saya memiliki dua kandidat yang akan digunakan pada buku ini, antara LunarVim dan AstroNvim. Pada akhirnya saya memilih AstroNvim karena ia memiliki proses pemasangan yang lebih sederhana dibanding yang lain yang pernah saya coba. Pada bagian berikutnya kita akan memulai pemasangan konfigurasi AstroNvim pada Neovim kita!

# Memasang AstroNvim

Pemasangan AstroNvim memerlukan beberapa syarat yang harus dipenuhi sebelum dapat menggunakannya. Syarat-syarat tersebut adalah:

- Nerd Fonts: AstroNvim membutuhkan *font* yang disebut Nerd Fonts untuk menampilkan *icon* pada *statusline* dan tombol navigasi.
- Neovim 0.8 (Tidak termasuk versi *nightly*): AstroNvim hanya dapat digunakan pada versi 0.8 dari Neovim. Jika kamu menggunakan versi yang lebih baru atau lebih lama dari Neovim, mungkin kamu tidak dapat menggunakan AstroNvim.
- Terminal dengan dukungan *true color*: AstroNvim memerlukan terminal yang mendukung *true color* untuk menampilkan warna yang tepat pada tema *default*. Jika kamu menggunakan tema lain, maka kebutuhan terhadap dukungan *true color* bisa bervariasi.

Selain syarat-syarat tersebut, AstroNvim juga memiliki beberapa fitur opsional yang memerlukan ketergantungan tambahan untuk dapat digunakan. Syarat-syarat opsional tersebut adalah:

- *ripgrep*: Memungkinkan kita untuk melakukan pencarian teks secara *real-time* di proyek kita menggunakan tombol tombol navigasi yang disediakan oleh AstroNvim.
- *lazygit*: Memungkinkan kita untuk menampilkan antarmuka Git dalam *terminal* dan melakukan aksi Git seperti *commit*, *push*, dan lainnya dengan mudah.
- *gdu*: Memungkinkan kita untuk menampilkan penggunaan *disk* dalam *terminal* dan memantau kapasitas penyimpanan kita dengan mudah.
- *bottom*: Memungkinkan kita untuk menampilkan daftar proses yang sedang berjalan dalam terminal dan melakukan aksi seperti mengakhiri proses dengan mudah.

- Python: Memungkinkan kita untuk menampilkan *interpreter* Python dalam *terminal* dan mengeksekusi perintah Python secara interaktif. Ini berguna untuk menguji kode Python kita tanpa perlu keluar dari Neovim.
- Node: Memungkinkan kita untuk menampilkan *interpreter* Node dalam *terminal* dan mengeksekusi perintah Node secara interaktif. Ini berguna untuk menguji kode Node kita tanpa perlu keluar dari Neovim.

Dengan memenuhi syarat-syarat dan persyaratan opsional tersebut, kita dapat melakukan pemasangan AstroNvim dan mulai menggunakannya untuk membantu kita dalam pengembangan proyek di Neovim.

Kendati opsional, tetapi pada bagian ini kita akan memasang seluruhnya pada sistem operasi yang kita gunakan. Hal ini untuk mendapatkan fitur penuh dari AstroNvim dan meminimalisir *error* yang tidak terduga.

AstroNvim bergantung pada beberapa *plugin* yang sudah disebutkan sebelumnya, sangat memungkinkan bila beberapa di antara ketergantungan tersebut membutuhkan syarat-syarat di atas. Misal *plugin* Mason.nvim yang digunakan oleh AstroNvim, *plugin* tersebut membutuhkan *package manager* eksternal seperti Cargo atau NPM yang di mana alat tersebut tersedia pada Node.

Seperti pada bagian-bagian pemasang sebelumnya, memasang alat-alat yang disyaratkan oleh AstroNvim pada sistem operasi yang berbeda membutuhkan langkah-langkah yang sesuai. Untuk itu, pada buku ini hanya mencakup langkah-langkah untuk sistem operasi macOS, Windows, dan Linux Ubuntu.

Membuat langkah-langkah yang rinci untuk sistem operasi Unix-like seperti Linux akan terlalu luas jangkauannya, mengingat Linux memiliki banyak sekali distribusi yang berbeda-beda. Masing-masing distro memiliki *package manager* yang berbeda, sehingga akan membuat langkah-langkahnya pun berbeda dari yang lain.

Kamu dapat menggunakan langkah-langkah yang ada sebagai referensi, karena secara garis besar seharusnya tidak jauh berbeda, hanya alat yang digunakan saja yang berbeda.

## macOS

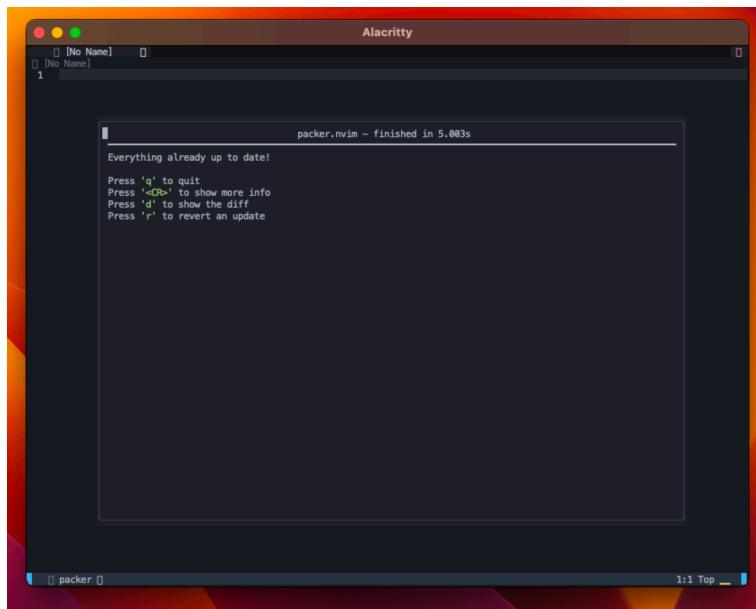
Pertama-tama, pastikan Git sudah terpasang. Pada dasarnya Git sudah terpasang pada saat kita memasang Xcode, jika tidak ada, kamu dapat memasangnya dengan perintah `brew install git`. Selain itu, kita perlu memasang beberapa ketergantungan yang lain dan memasang AstroNvim dengan langkah-langkah berikut:

1. Pastikan *program* Terminal terbuka, dan jalankan perintah berikut untuk memasang semua ketergantungan sekaligus:  
`brew install node python3 bottom gdu lazygit ripgrep`
2. Karena *terminal emulator* bawaan macOS tidak mendukung *true color*, kita perlu menggunakan alternatifnya seperti iTerm2 atau yang lain, seperti Alacritty. Pergi ke halaman rilis terbaru Alacritty (<https://github.com/alacritty/alacritty/releases/latest>) dan unduh paket yang sesuai dengan sistem operasi kamu, kemudian pasang seperti pada umumnya

▼ Assets 14		
<a href="#">Alacritty-msg.1.gz</a>	833 Bytes	Oct 13
<a href="#">Alacritty-v0.11.0-installer.msi</a>	2.15 MB	Oct 14
<a href="#">Alacritty-v0.11.0-portable.exe</a>	4.35 MB	Oct 13
<a href="#">Alacritty-v0.11.0.dmg</a>	5.05 MB	Oct 13

Daftar paket Alacritty

3. Jika sudah, buka Alacritty dan unduh konfigurasi AstroNvim dari GitHub menggunakan perintah `git clone https://github.com/AstroNvim/AstroNvim ~./config/nvim`. Perintah tersebut akan mengunduh *repository* AstroNvim dan menaruhnya di `~/config/nvim`
4. Jalankan perintah `nvim +PackerSync` untuk mulai memasang konfigurasi AstroNvim, tunggu hingga selesai



Memasang *plugin-plugin* ketergantungan

5. Jika terdapat *error*, abaikan saja dahulu dan tunggu hingga sisanya selesai terpasang, kita dapat memulai kembali proses pemasangan *plugin-plugin* yang tidak terpasang sebelumnya dengan perintah :PackerSync di dalam Neovim.
6. Apabila semuanya sudah berhasil, kita dapat keluar dari Neovim dan membukanya lagi dengan perintah nvim.



Neovim dengan konfigurasi AstroNvim

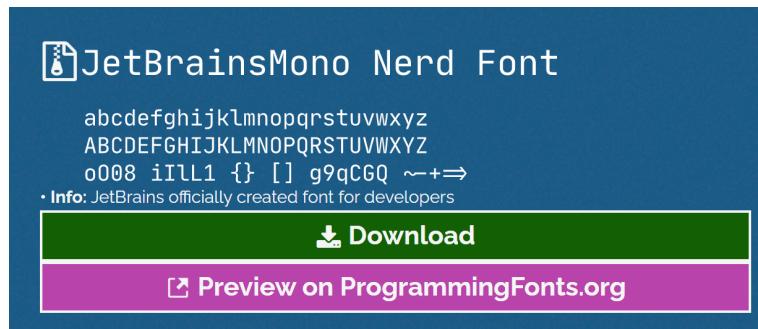
Sejauh ini kita sudah memasang konfigurasi AstroNvim, namun *icon* tidak muncul, ini dikarenakan *font* pada Gnome Terminal saat ini tidak dapat mengenalinya, untuk itu kita perlu memasang Nerd Fonts dan menggunakan di *terminal emulator* tersebut.

## Memasang Nerd Fonts

Kita membutuhkan Nerd Fonts untuk memunculkan *glyph icon* pada *terminal emulator* khususnya pada Neovim, karena AstroNvim menggunakan beberapa *plugin* yang di dalamnya menggunakan *icon-icon* yang dapat dibaca oleh Nerd Fonts. Bila kamu menggunakan WSL, kamu dapat mengikuti langkah-langkah pemasangan Nerd Fonts untuk sistem operasi Windows.

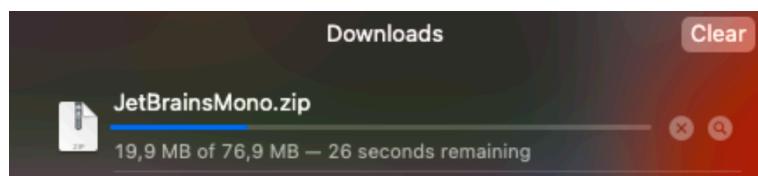
Berikut langkah-langkah pemasangan Nerd Fonts di Ubuntu:

1. Pergi ke situs web Nerd Fonts (<https://www.nerdfonts.com/font-downloads>)
2. Katakanlah font favoritmu adalah *JetBrainsMono*, maka klik tombol *Download* untuk mengunduh font tersebut



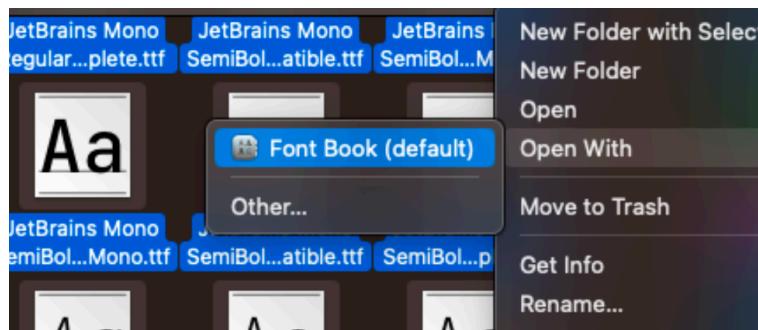
Mengunduh *font* JetBrainsMono

3. Tunggu hingga proses unduhan selesai



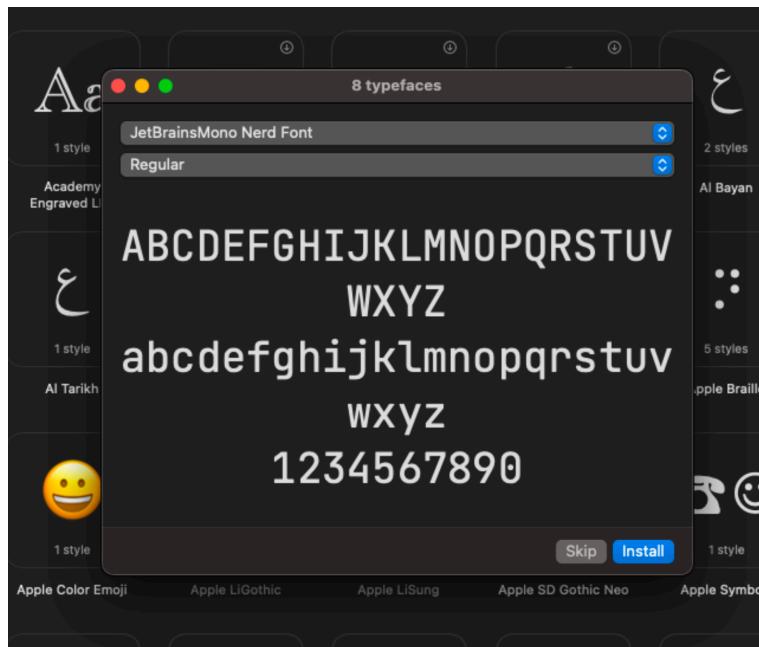
Proses unduhan *font*

4. Setelah selesai, ekstrak paket *.zip* tersebut ke dalam sebuah folder
5. Seleksi semua berkas *font*, klik kanan dan pilih *Open With > Font Book*



Menyeleksi semua berkas *font*

6. Akan terbuka aplikasi Font Book, klik *Install* pada semua *font*



Memasang semua *font*

7. Atur Alacritty agar menggunakan *font JetBrainsMono NF* dengan cara buat berkas `~/.alacritty.yml` dengan konfigurasi seperti ini:

`font:`

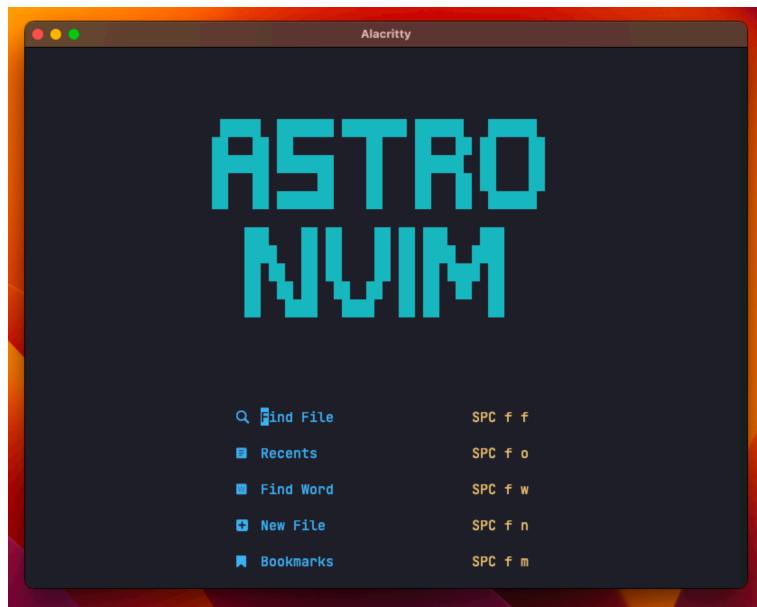
`normal:`

`family: JetBrainsMono NF`

`style: Regular`

`size: 15`

8. Tutup Alacritty dan buka kembali, buka *program* Neovim dengan perintah `nvim`. Seharusnya *icon* sudah muncul dengan cantiknya

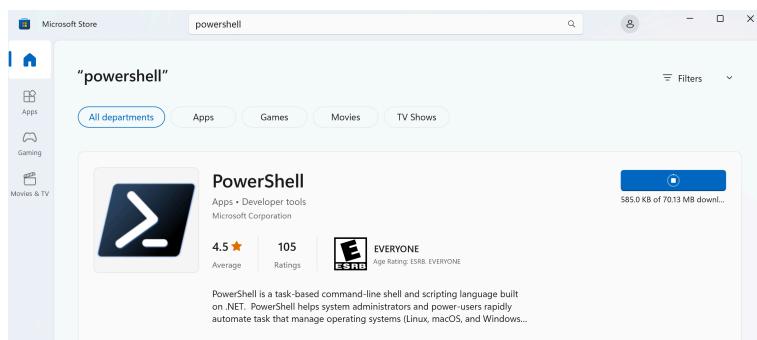


Neovim dengan konfigurasi AstroNvim dan JetBrainsMono

## Windows

Sebelumnya kita sudah memasang Scoop pada Windows untuk memasang paket-paket ketergantungan yang kita butuhkan, berikutnya kita butuh PowerShell 7 untuk memasang beberapa hal yang kita perlukan kedepannya.

Untuk memasang PowerShell 7, buka Microsoft Store, cari "*PowerShell*", dan klik *Get*



Mencari dan memasang PowerShell

Tunggu hingga pemasangan selesai dan klik *Open* untuk membuka *program* tersebut.

Sebelumnya sudah memasang Git, kita tidak perlu memasangnya lagi saat memasang konfigurasi AstroNvim dan beberapa ketergantungannya.

Berikut ini langkah-langkah untuk memasang konfigurasi AstroNvim dan ketergantungannya:

1. Pastikan berada di dalam PowerShell 7 dan jalankan perintah berikut untuk memasang semua ketergantungan AstroNvim sekaligus: `scoop install mingw nodejs python bottom gdu lazygit ripgrep`

```
PowerShell
Persisting bin
Persisting cache
Running pre_install script...
'python' (3.11.0) was installed successfully!
Installing 'python' (3.11.0) [arm64] from main bucket
Loading python-3.11.0-arm64.exe from cache
Checking hash of python-3.11.0-arm64.exe ... ok.
Running pre_install script...
Running installed script...
Linking ~\scoop\apps\python\current => ~\scoop\apps\python\3.11.0
Creating shim for 'python3'.
Creating shim for 'idle'.
Creating shim for 'idle3'.
Persisting Scripts
Persisting lib\site-packages
Running post_install script...
'python' (3.11.0) was installed successfully!
Notes
-----
Allow applications and third-party installers to find python by running:
"C:\Users\mhdnaivalazhar\scoop\apps\python\current\install-pep-514.reg"
Installing 'ripgrep' (13.0.0) [64bit] from main bucket
Loading ripgrep-13.0.0-x86_64-pc-windows-msvc.zip from cache
Checking hash of ripgrep-13.0.0-x86_64-pc-windows-msvc.zip ... ok.
Extracting ripgrep-13.0.0-x86_64-pc-windows-msvc.zip ...
Linking ~\scoop\apps\ripgrep\current => ~\scoop\apps\ripgrep\13.0.0
Creating shim for 'rg'.
'ripgrep' (13.0.0) was installed successfully!
PS C:\Users\mhdnaivalazhar>
```

Memasang beberapa ketergantungan sekaligus melalui Scoop

2. Unduh konfigurasi AstroNvim dari GitHub menggunakan perintah `git clone https://github.com/AstroNvim/AstroNvim $env:LOCALAPPDATA\nvim`. Perintah tersebut akan mengunduh *repository* AstroNvim dan menaruhnya di `C:\Users\{username}\AppData\Local\nvim`

```

PS C:\Users\mhdnaulazhar> git clone https://github.com/AstroNvim/AstroNvim $env:LOCALAPPDATA\nvim
Cloning into 'C:\Users\mhdnaulazhar\AppData\Local\nvim'...
remote: Enumerating objects: 6011, done.
remote: Counting objects: 100% (401/401), done.
remote: Compressing objects: 100% (200/200), done.
remote: Writing objects: 100% (6011/6011), done.
   100% (6011/6011) 1.76 MiB | 1.21 MiB/s, done.
Resolving deltas: 100% (3859/3859), done.
PS C:\Users\mhdnaulazhar>

```

Unduh konfigurasi AstroNvim dengan Git

3. Berikutnya buka *program* Neovim di dalam PowerShell dengan perintah `nvim`, seharusnya Neovim mulai memasang semua *plugin-plugin* yang digunakan oleh konfigurasi AstroNvim melalui *package manager* Packer

```

[packer] 1

[packer] packer.nvim - syncing 35 / 47 plugins
[packer] [progress] Please wait while plugins are installed...
[packer] 1,1 Top

```

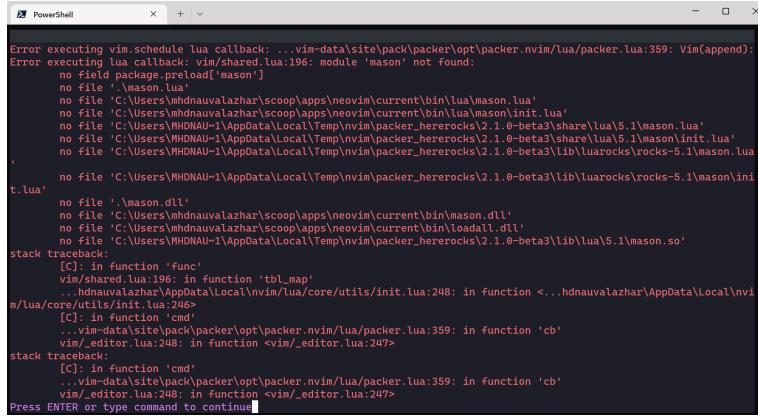
The output log shows:

- wbthomason/packer.nvim: checking current branch...
- x williamboan/mason: failed to install
- akinsho/bufferline.nvim: Cloning into 'C:\Users\mhdnaulazhar\AppData\Local\nvim-data\site\pack\folke\bufferline.nvim'...
- max3905/nvim-treesitter: Cloning into 'C:\Users\mhdnaulazhar\AppData\Local\nvim-data\site\pack\folke\whitespace.nvim'...
- hrsh7th/cmp-buffer: Cloning into 'C:\Users\mhdnaulazhar\AppData\Local\nvim-data\site\pack\hrsh7th\cmp-buffer'...
- williamboan/mason-lspconfig.nvim: Cloning into 'C:\Users\mhdnaulazhar\AppData\Local\nvim-data\site\pack\williamboan/mason-lspconfig.nvim'...
- x williamboan/mason-path: failed to install
- jose-elias-alvarez/null-ls.nvim: Cloning into 'C:\Users\mhdnaulazhar\AppData\Local\nvim-data\site\pack\p00f\nvim-ts-rainbow'...
- p00f\nvim-ts-rainbow: Cloning into 'C:\Users\mhdnaulazhar\AppData\Local\nvim-data\site\pack\p00f\nvim-ts-rainbow'...
- s1n7ax/nvim-window-picker: Cloning into 'C:\Users\mhdnaulazhar\AppData\Local\nvim-data\site\hrsh7th/nvim-cmp'...
- hrsh7th/nvim-cmp: Cloning into 'C:\Users\mhdnaulazhar\AppData\Local\nvim-data\site\pack\hrsh7th/nvim-cmp'...
- mrjones2014/smart-splits.nvim: Cloning into 'C:\Users\mhdnaulazhar\AppData\Local\nvim-data\familiu\ufdfelete.nvim': failed to install
- newbee-labs/spack.nvim: Cloning into 'C:\Users\mhdnaulazhar\AppData\Local\nvim-data\site\pack\newbee-labs/spack.nvim'...
- mvis-lua/plenary.nvim: Cloning into 'C:\Users\mhdnaulazhar\AppData\Local\nvim-data\site\pack\jayps21/mason-null-ls.nvim': failed to install
- x akinsho/toggleterm.nvim: Cloning into 'C:\Users\mhdnaulazhar\AppData\Local\nvim-data\site\pack\akinsho/toggleterm.nvim'...

Neovim memasang *plugin-plugin* melalui Packer

4. Seandainya terdapat *plugin* yang gagal dipasang, abaikan saja dahulu, kita dapat mencoba memasangnya kembali berikutnya. Seharusnya proses pemasangan berjalan lancar, namun bila terdapat pesan *error* yang menunjukkan sebuah *module* tidak ditemukan, biarkan juga, karena kemungkinan *plugin*

atau *module* tersebut belum terpasang namun Neovim mencoba menjalankannya. Tekan tombol ENTER hingga pesan tersebut hilang



```
Error executing vim.schedule lua callback: ...vim-data\site\pack\packer\opt\packer.nvim\lua\packer.lua:359: Vim(append):
Error executing lua callback: vim\shared.lua:196: module 'mason' not found:
    no field preloaded
    no file 'C:\Users\hdnaualazhar\scoop\apps\neovim\current\bin\lua\mason.lua'
    no file 'C:\Users\hdnaualazhar\scoop\apps\neovim\current\bin\lua\mason\init.lua'
    no file 'C:\Users\WHDNAU-1\AppData\Local\Temp\nvim\packer_hererocks\2.1.0-beta3\share\lua\5.1\mason.lua'
    no file 'C:\Users\WHDNAU-1\AppData\Local\Temp\nvim\packer_hererocks\2.1.0-beta3\share\lua\5.1\mason\init.lua'
    no file 'C:\Users\WHDNAU-1\AppData\Local\Temp\nvim\packer_hererocks\2.1.0-beta3\lib\luarocks\rocks-5.1\mason.lua'
    no file 'C:\Users\WHDNAU-1\AppData\Local\Temp\nvim\packer_hererocks\2.1.0-beta3\lib\luarocks\rocks-5.1\mason\ini
t.lua'
    no file './mason.dll'
    no file 'C:\Users\hdnaualazhar\scoop\apps\neovim\current\bin\mason.dll'
    no file 'C:\Users\hdnaualazhar\scoop\apps\neovim\current\bin\loadall.dll'
    no file 'C:\Users\WHDNAU-1\AppData\Local\Temp\nvim\packer_hererocks\2.1.0-beta3\lib\lua\5.1\mason.so'
stack traceback:
[C]: in function 'func'
  vim/shared.lua:196: in function 'tbl_map'
...hdnaualazhar\AppData\Local\nvim\lua\core\utils/init.lua:248: in function <...hdnaualazhar\AppData\Local\nvi
m\lua\core\utils/init.lua:246>
[C]: in function 'cmd'
  ...vim-data\site\pack\packer\opt\packer.nvim\lua\packer.lua:359: in function 'cb'
  vim/_editor.lua:248: in function <vim/_editor.lua:247>
stack traceback:
[C]: in function 'cmd'
  ...vim-data\site\pack\packer\opt\packer.nvim\lua\packer.lua:359: in function 'cb'
  vim/_editor.lua:248: in function <vim/_editor.lua:247>
Press ENTER or type command to continue
```

Pesan error Neovim mengenai *module* tidak ditemukan

5. Jika Neovim tidak lagi memasang *plugin* seperti sebelumnya, kita dapat keluar dari *program* Neovim dan membukanya lagi, gunakan perintah :q untuk keluar dari Neovim dan buka lagi dengan perintah nvim. Seharusnya Neovim sudah berhasil menggunakan konfigurasi dari AstroNvim



Tampilan awal Neovim dengan konfigurasi AstroNvim

- Kendati sudah terlihat bagus, namun proses pemasanganan belum selesai, jalankan kembali *package manager* agar memasang *plugin-plugin* yang sebelumnya gagal terpasang, gunakan perintah :PackerSync untuk melakukannya

```
PowerShell
[+] packer.nvim - syncing 47 / 47 plugins
  nvim-treesitter\nvim-treesitter: checking current branch...
  ○ mrjones2014\smart-splits.nvim: remote: Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
  ○ wbthomason\packer.nvim: remote: Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
  ○ hrsh7th\nvim-cmp: checking current branch...
  ○ nvim-lua-plenary.nvim: checking current branch...
  ○ rcarriga\ nvim-notify: remote: Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
  ○ nvim-telescope\telescope.nvim: checking current branch...
  ○ akindo\foligoterm.nvim: checking current branch...
  ○ nvim-whichkey\whichkey.nvim: checking current branch...
  ○ jose-elias-alvarez\nvim-null-ls: remote: Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
  ○ max307579\better-escape.nvim: remote: Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
  ○ neovim\nvim-lspconfig: checking current branch...
  ○ p0f\fnvim-ts-rainbow: checking current branch...
  ○ goolord\alpha-nvim: checking current branch...
  ○ lewis6991\impatient.nvim: checking current branch...
  ○ rebelot\neirline.nvim: checking out a94399e...
  ○ sin7ax\nvim-window-picker: checking current branch...
  ○ L3MON4D3\LuaSnip: checking out 3fa5c8d...

```

SPC f n  
SPC f m

Memasang kembali *plugin-plugin* yang belum terpasang

- Sama seperti sebelumnya, bila terdapat *plugin* yang gagal dipasang, kita dapat mengabaikannya terlebih dahulu hingga proses pemasangan *plugin* saat ini selesai, karena kita dapat mencoba lagi setelahnya. Bila proses pemasangan *plugin* tidak ada *error*, maka akan tampil seperti berikut:

```
PowerShell
[+] packer.nvim - finished in 65.610s
✓ Installed hrsh7th\cmp-path
✓ Installed famu\bufdelete.nvim
✓ Installed jvns\qflist-aspect\ll-1s.nvim
✓ Installed williambonan\mason.nvim
✓ Installed MunifTanjim\mru.nvim
✓ Installed stevearc\dressing.nvim
✓ Installed windwp\fnvim-ts-autotag
✓ Installed numToStr\Comment.nvim

Press 'q' to quit
Press <CR> to show more info
Press 'd' to show the diff
Press 'r' to revert an update


```

SPC f n  
SPC f m

Pemasangan *plugin* Neovim berhasil

- Gunakan perintah q untuk keluar dari *floating window* Packer

Sejauh ini kita sudah memasang konfigurasi AstroNvim, namun *icon* tidak muncul, ini dikarenakan *font* pada Gnome Terminal saat ini tidak dapat mengenalinya, untuk itu kita perlu memasang Nerd Fonts dan menggunakan di *terminal emulator* tersebut.

## Memasang Nerd Fonts

Kita membutuhkan Nerd Fonts untuk memunculkan *glyph icon* pada *terminal emulator* khususnya pada Neovim, karena AstroNvim menggunakan beberapa *plugin* yang di dalamnya menggunakan *icon-icon* yang dapat dibaca oleh Nerd Fonts.

Langkah-langkah berikut ini juga berlaku untuk pengguna WSL, karena bagaimanapun *program terminal* yang menjalankan WSL berjalan di dalam lingkungan Windows.

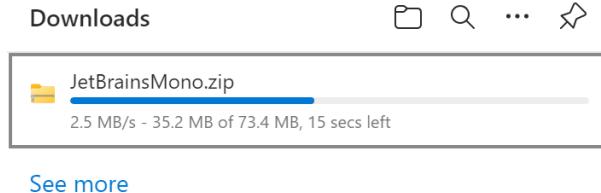
Berikut langkah-langkah memasang Nerd Fonts di Windows:

1. Pergi ke situs web Nerd Fonts (<https://www.nerdfonts.com/font-downloads>)
2. Katakanlah *font* favoritmu adalah *JetBrainsMono*, maka klik tombol *Download* untuk mengunduh *font* tersebut



Mengunduh *font* JetBrainsMono

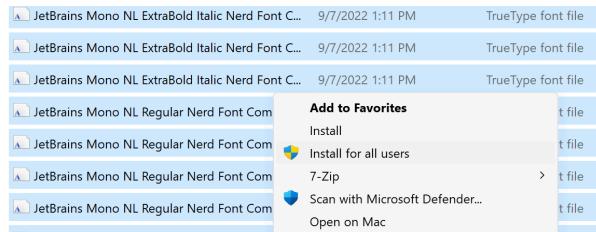
3. Tunggu hingga proses unduhan selesai



See more

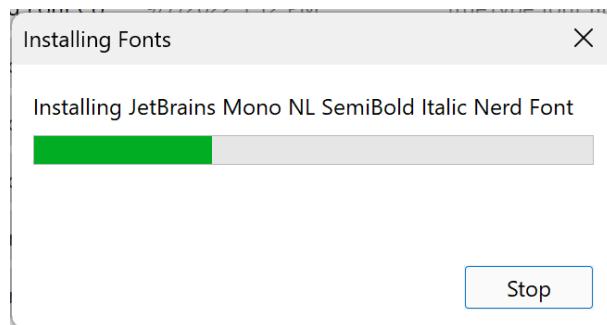
#### Proses unduhan *font*

4. Setelah selesai, ekstrak paket *.zip* tersebut ke dalam sebuah folder
5. Seleksi semua berkas *font* kecuali berkas *readme.md*, klik kanan, pilih *Show more options* dan pilih *Install for all users*



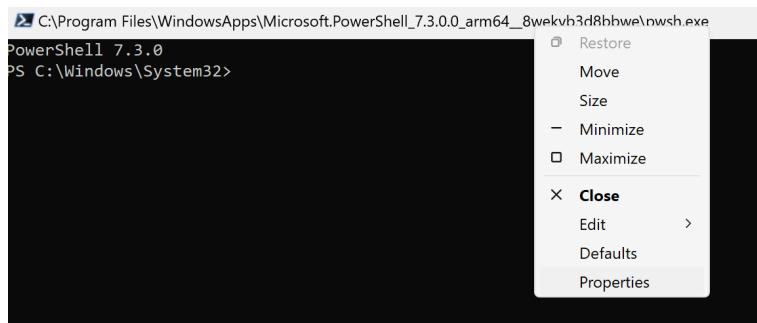
Memilih semua berkas *font* dan memasangnya

6. Tunggu hingga proses pemasangan *font* selesai



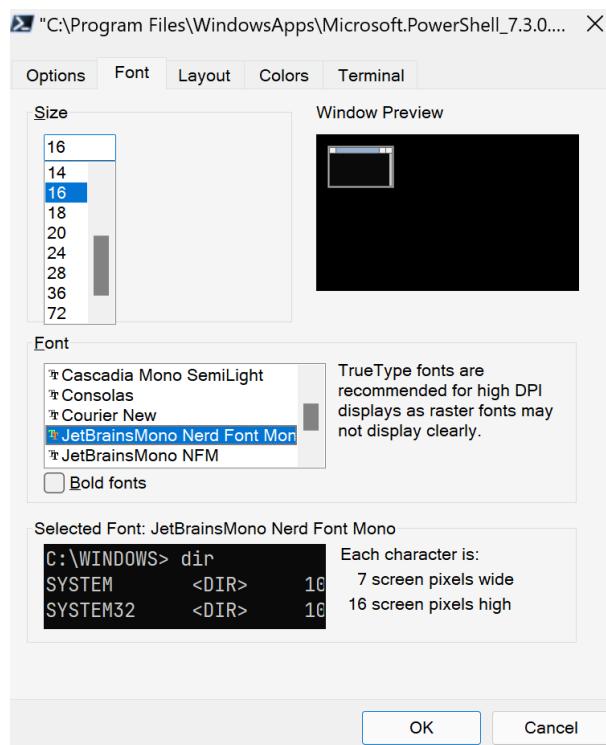
Proses pemasangan *font*

7. Tutup *program PowerShell* bila sedang terbuka dan buka kembali, klik kanan pada bagian judul aplikasi *PowerShell* dan pilih *Properties*



Membuka jendela *Properties* PowerShell

#### 8. Cari *JetBrainsMono Nerd Font Mono* dan klik tombol *OK*



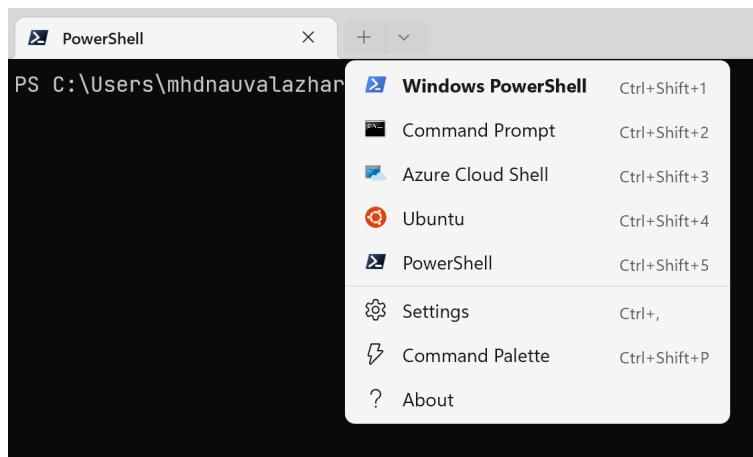
Memilih *JetBrainsMono Nerd Font Mono* sebagai font PowerShell

#### 9. Buka kembali *program Neovim*, dan seharusnya sekarang *icon* sudah muncul di dalam Neovim



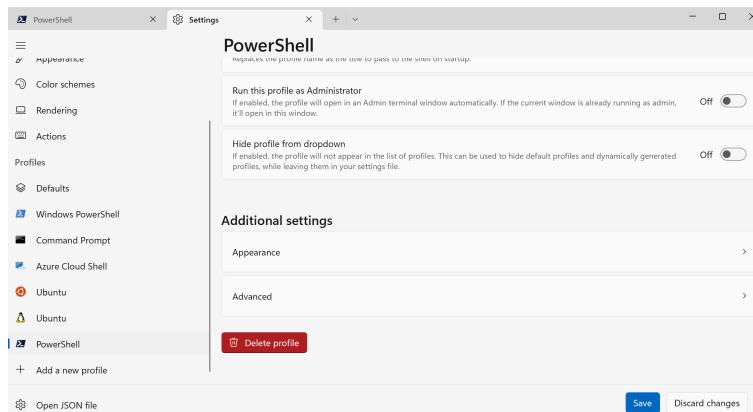
Neovim dengan *icon* dari Nerd Fonts

10. Seandainya kamu menggunakan aplikasi Terminal di Windows, kamu dapat mengubah *font* pada pengaturan, klik tombol dengan *icon chevron* di samping *tab*, dan pilih *Settings*



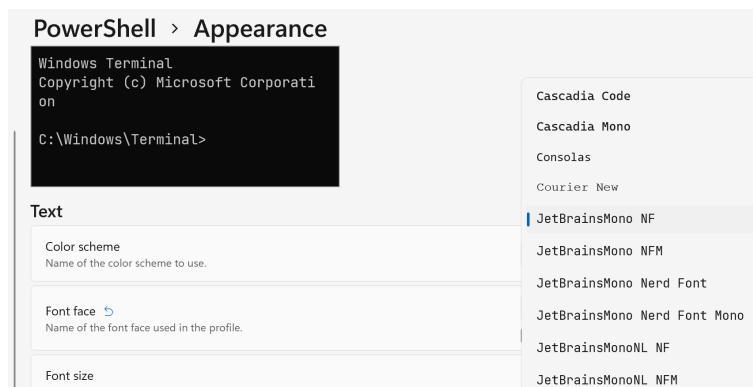
Membuka jendela *Settings* aplikasi Terminal

11. Pilih PowerShell di bagian *sidebar*, klik pengaturan *Appearance* yang berada di bawah



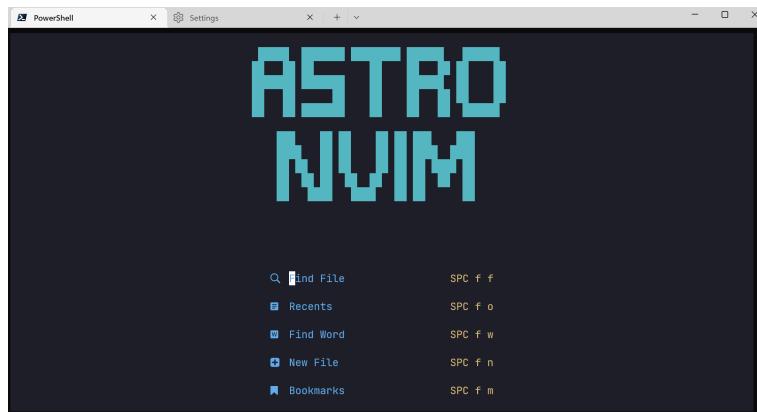
Memilih PowerShell dan menu *Appearance*

## 12. Pilih *JetBrainsMono NF* pada pengaturan *Font face*



Memilih *JetBrainsMono NF* pada pengaturan *Font face*

## 13. Klik tombol *Save* untuk menyimpannya, kembali ke *tab PowerShell* dan buka *program* Neovim dengan perintah `nvim`. Seharusnya *icon* sudah muncul dengan cantiknya



Neovim dengan *icon* di dalam PowerShell Terminal

Sekarang semua *icon* di dalam Neovim seharusnya muncul. Nerd Fonts tidak hanya untuk Neovim, juga untuk *program* Terminal secara menyeluruh, kita mungkin menggunakan *tool CLI* yang menampilkan *icon* dari Nerd Font, maka tidak perlu khawatir lagi, *icon* tersebut seharusnya akan muncul.

## Unix-like

Seperti yang sudah dijelaskan sebelumnya, saya hanya membuat langkah-langkah untuk Linux Ubuntu saja. Untuk sistem operasi atau distribusi Linux jenis lain, dapat mengikuti langkah-langkah berikut sebagai referensi namun dengan caranya masing-masing.

Berikut memasang AstroNvim di Ubuntu:

1. Pastikan *program* Terminal terbuka
2. Jalankan perintah `sudo apt install git` untuk memasang Git

```

parallels@ubuntu-linux-20-04-desktop: ~
Get:1 http://ports.ubuntu.com/ubuntu-ports focal/main arm64 liberror-perl all 0.17029-1 [26.5 kB]
Get:2 http://ports.ubuntu.com/ubuntu-ports focal-updates/main arm64 git-man all 1:2.25.1-1ubuntu3.6 [887 kB]
Get:3 http://ports.ubuntu.com/ubuntu-ports focal-updates/main arm64 git arm64 1:2.25.1-1ubuntu3.6 [4,338 kB]
Fetched 5,251 kB in 8s (693 kB/s)
Selecting previously unselected package liberror-perl.
(Reading database ... 143393 files and directories currently installed.)
Preparing to unpack .../liberror-perl_0.17029-1_all.deb ...
Unpacking liberror-perl (0.17029-1) ...
Selecting previously unselected package git-man.
Preparing to unpack .../git-man_1%3a2.25.1-1ubuntu3.6_all.deb ...
Unpacking git-man (1:2.25.1-1ubuntu3.6) ...
Selecting previously unselected package git.
Preparing to unpack .../git_1%3a2.25.1-1ubuntu3.6_arm64.deb ...
Unpacking git (1:2.25.1-1ubuntu3.6) ...
Setting up liberror-perl (0.17029-1) ...
Setting up git-man (1:2.25.1-1ubuntu3.6) ...
Setting up git (1:2.25.1-1ubuntu3.6) ...
Processing triggers for man-db (2.9.1-1) ...
parallels@ubuntu-linux-20-04-desktop: ~

```

Memasang Git dengan APT

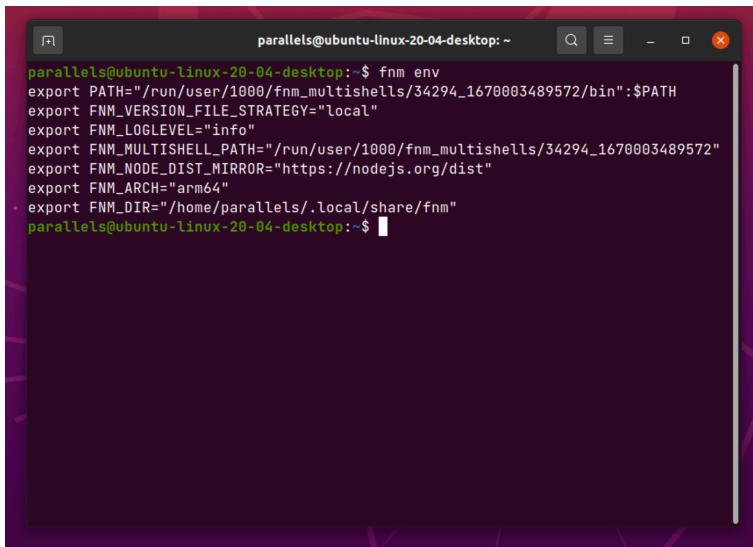
3. Setelah selesai, kita dapat memasang FNM (*tool* untuk mengelola versi Node JS) untuk memasang Node JS, pergi ke halaman rilis FNM (<https://github.com/Schniz/fnm/releases>) untuk mengunduh paket FNM yang sesuai dengan sistem operasi kamu

Asset	Size	Last updated
fnm-arm32.zip	3.81 MB	15 days ago
fnm-arm64.zip	4.07 MB	15 days ago
fnm-linux.zip	2.83 MB	15 days ago
fnm-macos.zip	2.59 MB	15 days ago
fnm-windows.zip	2.92 MB	15 days ago
Source code (zip)		15 days ago
Source code (tar.gz)		15 days ago

Daftar paket FNM

4. Ekstrak paket tersebut dengan perintah `unzip fnm-arm64.zip`, sesuaikan argumen nama berkas sesuai dengan nama paket yang diunduh, dalam hal ini saya mengunduh paket FNM untuk arsitektur *arm64*, maka nama berkasnya adalah *fnm-arm64*
5. Berikan mode *execution* untuk *program* FNM dengan perintah `chmod u+x fnm`

6. Pindahkan *program* FNM agar dapat diakses secara global dengan perintah `sudo mv fnm /usr/bin/fnm`
7. Kita sudah dapat menggunakan FNM, tapi belum selesai proses pemasangannya, kita perlu menjalankan perintah `fnm env`
8. Perintah tersebut akan mencetak kode yang perlu kita taruh di dalam `~/.bashrc`



```
parallels@ubuntu-linux-20-04-desktop:~$ fnm env
export PATH="/run/user/1000/fnm_multishells/34294_1670003489572/bin":$PATH
export FNM_VERSION_FILE_STRATEGY="local"
export FNM_LOGLEVEL="info"
export FNM_MULTISHELL_PATH="/run/user/1000/fnm_multishells/34294_1670003489572"
export FNM_NODE_DIST_MIRROR="https://nodejs.org/dist"
export FNM_ARCH="arm64"
export FNM_DIR="/home/parallels/.local/share/fnm"
parallels@ubuntu-linux-20-04-desktop:~$
```

Hasil perintah `fnm env`

9. Tambahkan ke baris terakhir berkas `~/.bashrc` dan jalankan perintah `source ~/.bashrc`
10. Sekarang kita dapat memasang Node JS melalui FNM dengan menggunakan perintah `fnm install v18`
11. Tunggu hingga selesai dan gunakan perintah `fnm use v18` untuk menggunakan versi Node tersebut
12. Berikutnya kita dapat memasang Python dengan perintah `sudo apt install python3.8`

```
Preparing to unpack .../libpython3.8-stdlib:arm64_3.8.10-0ubuntu1~20.04.5_arm64.deb ...
.
Unpacking libpython3.8-stdlib:arm64 (3.8.10-0ubuntu1~20.04.5) over (3.8.10-0ubuntu1~20.04.2) ...
Preparing to unpack .../python3.8-minimal_3.8.10-0ubuntu1~20.04.5_arm64.deb ...
Unpacking python3.8-minimal (3.8.10-0ubuntu1~20.04.5) over (3.8.10-0ubuntu1~20.04.0) ...
Preparing to unpack .../libpython3.8-minimal_3.8.10-0ubuntu1~20.04.5_arm64.deb ...
.
Unpacking libpython3.8-minimal:arm64 (3.8.10-0ubuntu1~20.04.5) over (3.8.10-0ubuntu1~20.04.2) ...
Setting up libpython3.8-minimal:arm64 (3.8.10-0ubuntu1~20.04.5) ...
Setting up python3.8-minimal (3.8.10-0ubuntu1~20.04.5) ...
Setting up libpython3.8-stdlib:arm64 (3.8.10-0ubuntu1~20.04.5) ...
Setting up python3.8 (3.8.10-0ubuntu1~20.04.5) ...
Setting up libpython3.8:arm64 (3.8.10-0ubuntu1~20.04.5) ...
Processing triggers for mime-support (3.64ubuntu1) ...
Processing triggers for gnome-menus (3.36.0-1ubuntu1) ...
Processing triggers for libc-bin (2.31-0ubuntu9.2) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for desktop-file-utils (0.24-1ubuntu3) ...
parallels@ubuntu-linux-20-04-desktop:~$
```

### Memasang Python melalui APT

13. Ketergantungan berikutnya yang perlu dipasang adalah GDU, pergi ke halaman rilis GDU (<https://github.com/dun-dee/gdu/releases>) dan unduh paket yang sesuai dengan sistem operasi kamu, dalam hal ini saya mengunduh paket *gdu]linux\_arm64.tgz*

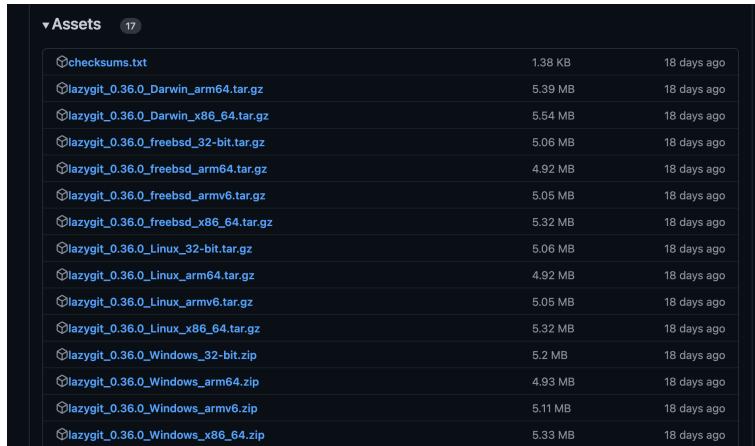
Assets	36
gdu-5.20.0-1.aarch64.rpm	2.8 MB
gdu-5.20.0-1.src.rpm	1.45 MB
gdu-5.20.0-1.x86_64.rpm	3.08 MB
gdu-5.20.0.tgz	4.74 MB
gdu.1.tgz	1.49 KB
gdu_android_arm64.tgz	3.42 MB
gdu_darwin_amd64.tgz	3.74 MB
gdu_darwin_arm64.tgz	3.56 MB
gdu_freebsd_386.tgz	3.42 MB
gdu_freebsd_amd64.tgz	3.6 MB
gdu_freebsd_arm.tgz	3.4 MB
gdu_linux_386.tgz	3.42 MB
gdu_linux_amd64.tgz	3.75 MB
gdu_linux_amd64_static.tgz	3.6 MB
gdu_linux_arm.tgz	3.4 MB
gdu_linux_arm64.tgz	3.27 MB

### Daftar paket GDU

14. Ekstrak paket tersebut dengan perintah:

```
tar -xf gdu_linux_arm64.tgz
```

15. Berikan mode *execution* untuk program GDU dengan perintah:  
`chmod u+x gdu_linux_arm64`
16. Agar GDU dapat diakses secara global, gunakan perintah `sudo mv gdu_linux_arm64 /usr/bin/gdu`
17. Berikutnya adalah Lazygit, pergi ke halaman rilis Lazygit (<https://github.com/jesseduffield/lazygit/releases>) dan unduh paket yang sesuai dengan sistem operasi kamu, dalam hal ini saya mengunduh paket `lazygit_0.36.0_Linux_arm64.tar.gz`

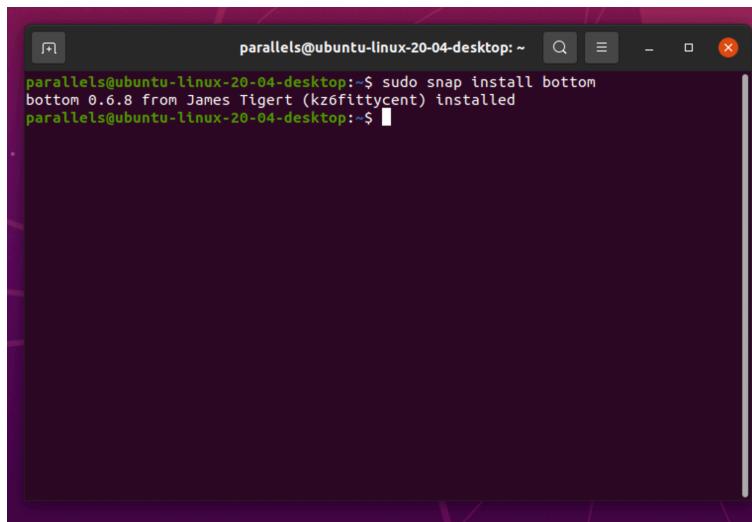


The screenshot shows a table of assets for the 'lazygit\_0.36.0\_Linux\_arm64.tar.gz' release. The columns are 'Assets' (with a count of 17), 'File' (with icons for each file type), 'Size', and 'Last modified'. The files listed are:

File	Size	Last modified
checksums.txt	1.38 KB	18 days ago
lazygit_0.36.0_Darwin_arm64.tar.gz	5.39 MB	18 days ago
lazygit_0.36.0_Darwin_x86_64.tar.gz	5.54 MB	18 days ago
lazygit_0.36.0_freebsd_32-bit.tar.gz	5.06 MB	18 days ago
lazygit_0.36.0_freebsd_arm64.tar.gz	4.92 MB	18 days ago
lazygit_0.36.0_freebsd_armv6.tar.gz	5.05 MB	18 days ago
lazygit_0.36.0_freebsd_x86_64.tar.gz	5.32 MB	18 days ago
lazygit_0.36.0_Linux_32-bit.tar.gz	5.06 MB	18 days ago
lazygit_0.36.0_Linux_arm64.tar.gz	4.92 MB	18 days ago
lazygit_0.36.0_Linux_armv6.tar.gz	5.05 MB	18 days ago
lazygit_0.36.0_Linux_x86_64.tar.gz	5.32 MB	18 days ago
lazygit_0.36.0_Windows_32-bit.zip	5.2 MB	18 days ago
lazygit_0.36.0_Windows_arm64.zip	4.93 MB	18 days ago
lazygit_0.36.0_Windows_armv6.zip	5.11 MB	18 days ago
lazygit_0.36.0_Windows_x86_64.tar.gz	5.33 MB	18 days ago

Daftar paket Lazygit

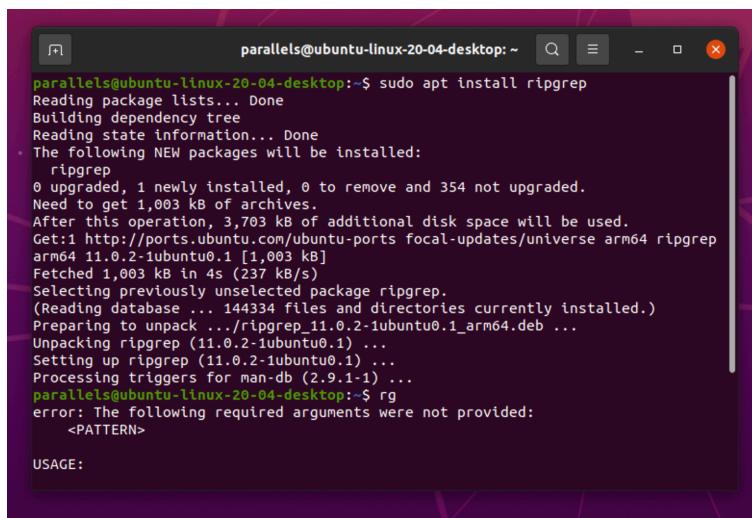
18. Ekstrak paket tersebut dengan perintah:  
`tar -xf lazygit_0.36.0_Linux_arm64.tar.gz`
19. Berikan mode *execution* dengan perintah `chmod u+x lazygit`
20. Agar Lazygit dapat diakses secara global, gunakan perintah `sudo mv lazygit /usr/bin/lazygit`
21. Berikutnya adalah Bottom, untuk memasang Bottom kita dapat menggunakan Snapcraft untuk memasangnya dengan perintah `sudo snap install bottom`



```
parallels@ubuntu-linux-20-04-desktop:~$ sudo snap install bottom
bottom 0.6.8 from James Tigert (kzofiftycent) installed
parallels@ubuntu-linux-20-04-desktop:~$
```

Memasang Bottom melalui Snapcraft

22. Terakhir adalah Ripgrep, kita dapat memasang Ripgrep melalui APT dengan perintah `sudo apt install ripgrep`

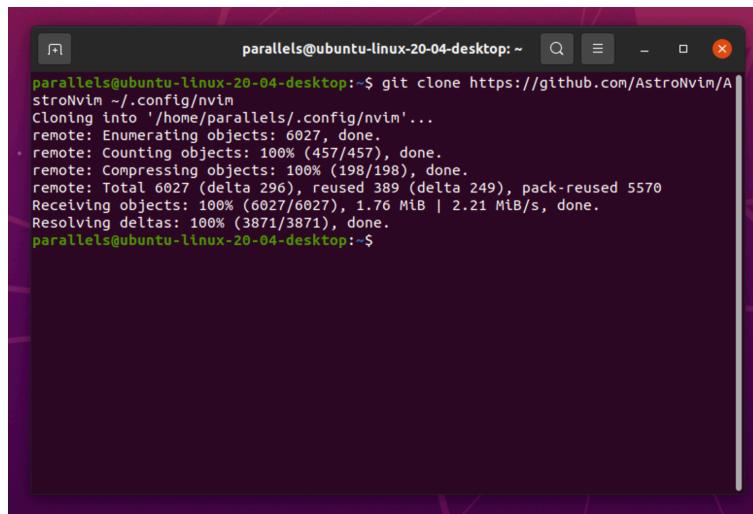


```
parallels@ubuntu-linux-20-04-desktop:~$ sudo apt install ripgrep
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  ripgrep
0 upgraded, 1 newly installed, 0 to remove and 354 not upgraded.
Need to get 1,003 kB of archives.
After this operation, 3,703 kB of additional disk space will be used.
Get:1 http://ports.ubuntu.com/ubuntu-ports focal-updates/universe arm64 ripgrep
arm64 11.0.2-1ubuntu0.1 [1,003 kB]
Fetched 1,003 kB in 4s (237 kB/s)
Selecting previously unselected package ripgrep.
(Reading database ... 144334 files and directories currently installed.)
Preparing to unpack .../ripgrep_11.0.2-1ubuntu0.1_arm64.deb ...
Unpacking ripgrep (11.0.2-1ubuntu0.1) ...
Setting up ripgrep (11.0.2-1ubuntu0.1) ...
Processing triggers for man-db (2.9.1-1) ...
parallels@ubuntu-linux-20-04-desktop:~$ rg
error: The following required arguments were not provided:
  <PATTERN>

USAGE:
```

Memasang Ripgrep melalui APT

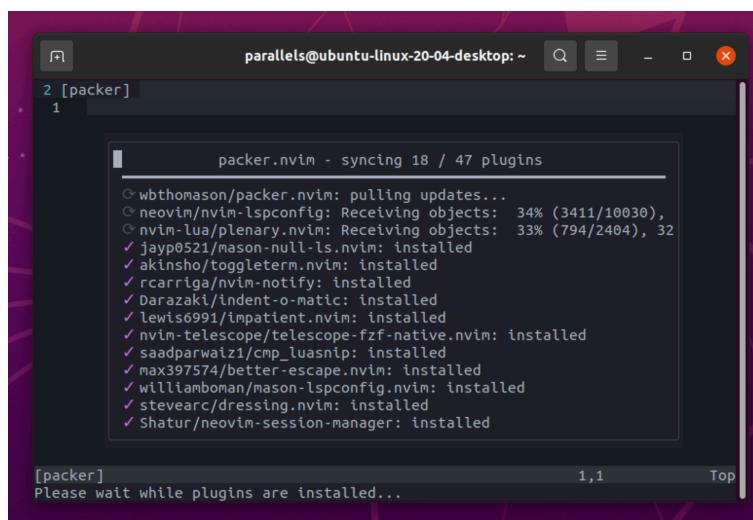
23. Setelah semua ketergantungan dipasang, saatnya memasang konfigurasi AstroNvim dengan Git menggunakan perintah `git clone https://github.com/AstroNvim/AstroNvim ~/.config/nvim`



```
parallels@ubuntu-linux-20-04-desktop:~$ git clone https://github.com/AstroNvim/AstroNvim ~/.config/nvim
Cloning into '/home/parallels/.config/nvim'...
remote: Enumerating objects: 6027, done.
remote: Counting objects: 100% (457/457), done.
remote: Compressing objects: 100% (198/198), done.
remote: Total 6027 (delta 296), reused 389 (delta 249), pack-reused 5570
Receiving objects: 100% (6027/6027), 1.76 MiB | 2.21 MiB/s, done.
Resolving deltas: 100% (3871/3871), done.
parallels@ubuntu-linux-20-04-desktop:~$
```

Mengunduh konfigurasi AstroNvim dengan Git

24. Setelah selesai mengunduh konfigurasi AstroNvim, buka *program* Neovim dengan perintah `nvim`. Saat terbuka, Neovim akan mulai menjalankan Packer untuk memasang semua ketergantungan *plugin-plugin* yang digunakan oleh konfigurasi AstroNvim



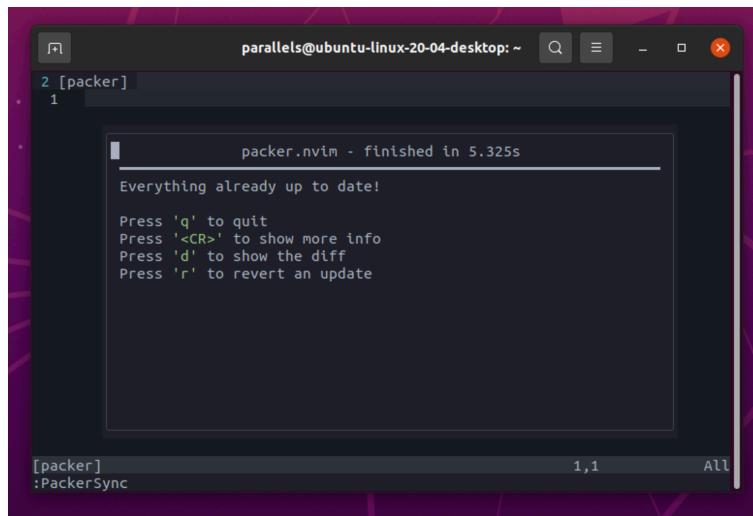
```
parallels@ubuntu-linux-20-04-desktop:~$ nvim
2 [packer]
1

[packer] packer.nvim - syncing 18 / 47 plugins
[packer] wbthomason/packer.nvim: pulling updates...
[packer] neovim/nvim-lspconfig: Receiving objects: 34% (3411/10030),
[packer] nvim-lua/plenary.nvim: Receiving objects: 33% (794/2404), 32
✓ jayp0521/mason-null-ls.nvim: installed
✓ akinsho/toggleterm.nvim: installed
✓ rcarriga/nvim-notify: installed
✓ Darazaki/indent-o-matic: installed
✓ lewis6991/impatient.nvim: installed
✓ nvim-telescope/telescope-fzf-native.nvim: installed
✓ saadparwaiz1/cmp_luasnip: installed
✓ max397574/better-escape.nvim: installed
✓ williamboman/mason-lspconfig.nvim: installed
✓ stevearc/dressing.nvim: installed
✓ Shatur/neovim-session-manager: installed

[packer]
Please wait while plugins are installed... 1,1 Top
```

Proses pemasangan *plugin-plugin*

25. Tunggu prosesnya hingga selesai



```
parallels@ubuntu-linux-20-04-desktop: ~
2 [packer]
1

[packer.nvim - finished in 5.325s]
Everything already up to date!

Press 'q' to quit
Press '<CR>' to show more info
Press 'd' to show the diff
Press 'r' to revert an update

[packer]
:PackerSync
1,1 All
```

Proses pemasangan sudah selesai

26. Jika terdapat *error*, abaikan saja dahulu dan tunggu hingga si-sanya selesai terpasang, kita dapat memulai kembali proses pemasangan *plugin-plugin* yang tidak terpasang sebelumnya dengan perintah :PackerSync di dalam Neovim
27. Apabila semuanya sudah berhasil, kita dapat keluar dari Neovim dan membukanya lagi dengan perintah nvim



Neovim dengan konfigurasi AstroNvim

Sejauh ini kita sudah memasang konfigurasi AstroNvim, namun *icon* tidak muncul, ini dikarenakan *font* pada Gnome Terminal saat ini tidak dapat mengenalinya, untuk itu kita perlu memasang Nerd Fonts dan menggunakan di *terminal emulator* tersebut.

## Memasang Nerd Fonts

Kita membutuhkan Nerd Fonts untuk memunculkan *glyph icon* pada *terminal emulator* khususnya pada Neovim, karena AstroNvim menggunakan beberapa *plugin* yang di dalamnya menggunakan *icon-icon* yang dapat dibaca oleh Nerd Fonts. Bila kamu menggunakan WSL, kamu dapat mengikuti langkah-langkah pemasangan Nerd Fonts untuk sistem operasi Windows.

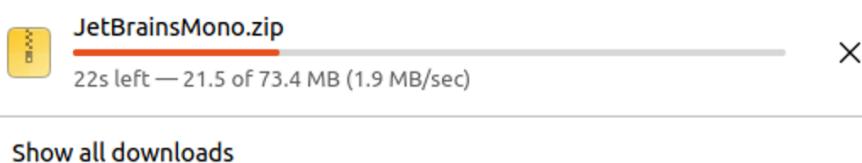
Berikut langkah-langkah pemasangan Nerd Fonts di Ubuntu:

1. Pergi ke situs web Nerd Fonts (<https://www.nerdfonts.com/font-downloads>)
2. Katakanlah *font* favoritmu adalah *JetBrainsMono*, maka klik tombol *Download* untuk mengunduh *font* tersebut



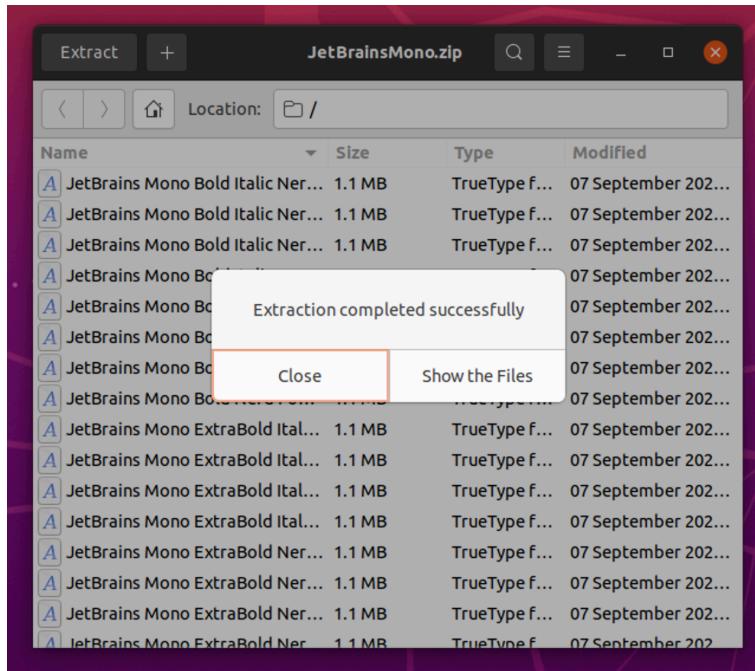
Mengunduh *font* JetBrainsMono

3. Tunggu hingga proses unduhan selesai



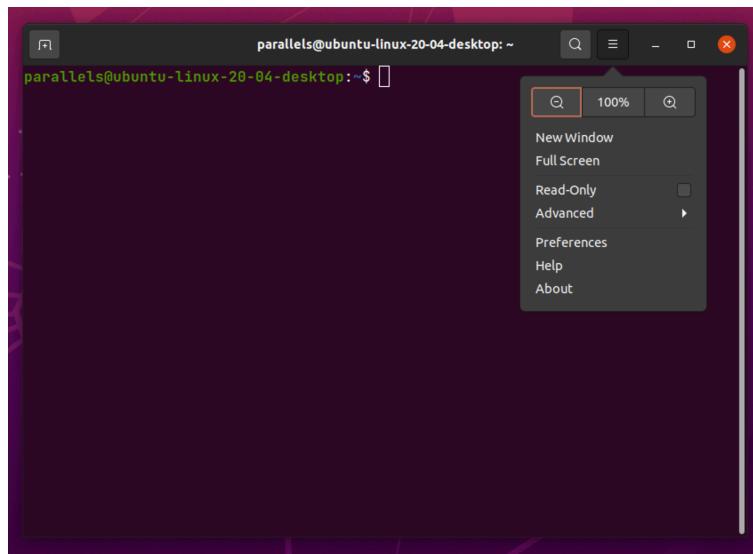
### Proses unduhan *font*

4. Berikutnya ekstrak paket *font* tersebut ke dalam sebuah folder, misal *JetBrains Mono*



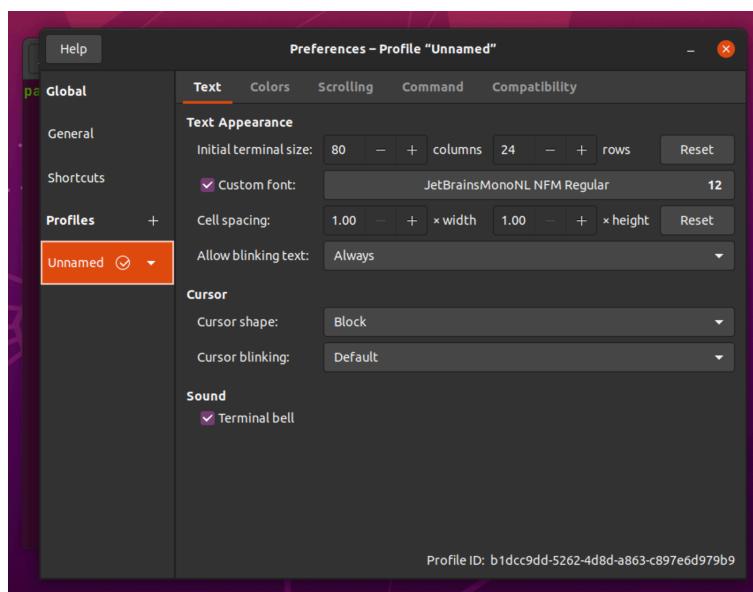
Berhasil mengekstrak paket *font*

5. Untuk memasang semua *font* tersebut, kita dapat menggunakan perintah `sudo mv JetBrains\ Mono/ /usr/local/share/fonts/`
6. Atur Gnome Terminal agar menggunakan *font JetBrainsMono* dengan cara klik *icon bar* pada bagian atas *terminal*, lalu pilih *Preferences*



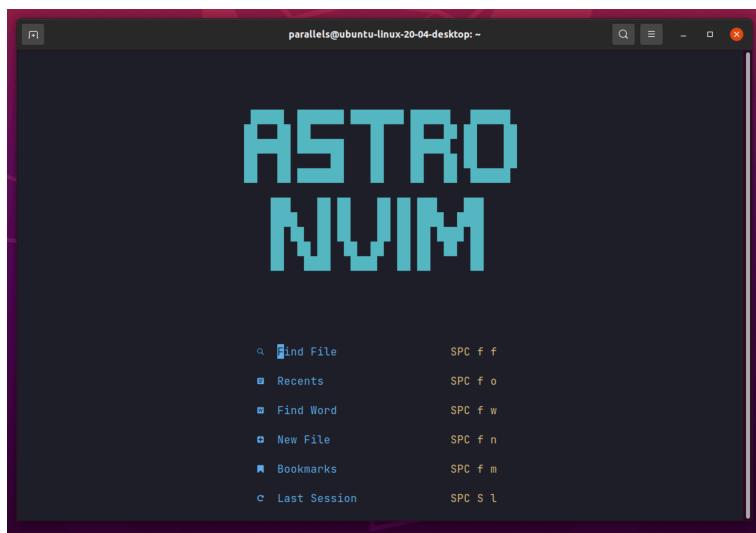
Menu Bar terminal

7. Klik pada *profile* di sebelah kiri, misal nama *profile*-nya adalah *Unnamed*, lalu ceklis pada *Custom font* dan pilih *JetBrainsMonoNL NFM Regular*



Memilih font *JetBrainsMono*

- Setelah itu buka kembali *program* Neovim dengan perintah `nvim`, seharusnya *icon* sudah muncul dengan cantiknya



Neovim dengan *icon*

# Penggunaan

Pada bagian ini, kita akan membahas tentang penggunaan AstroNvim sebagai *layer* IDE. AstroNvim merupakan sebuah *layer* yang dapat ditambahkan pada editor teks Neovim untuk meningkatkan fitur dan fungsi IDE-nya.

Untuk memulai, kita dapat menggunakan kombinasi *keyboard* atau perintah tertentu untuk mengakses fitur-fitur dasar AstroNvim. Misalnya, untuk membuka *terminal*, kita dapat menekan tombol F7. Selain itu, kita juga dapat menggunakan tombol SPASI-e untuk membuka *file explorer*.

AstroNvim juga menyediakan *default mappings* yang lainnya. Misal, kita dapat menggunakan perintah jj sebagai pengganti tombol ESCAPE atau perintah SPASI-ff untuk mencari berkas di dalam proyek saat ini menggunakan *plugin* Telescope.

Selain itu, AstroNvim juga menyediakan konfigurasi yang dapat kita kustomisasi sesuai dengan kebutuhan. Kita dapat mengubah konfigurasi seperti mengubah tema, mengubah konfigurasi *plugin* bawaan, dan lain-lain. Kita juga dapat mengelola LSP, debugger, formatter, dan linter agar sesuai dengan kebutuhan kita, pengelolaan ini dibantu dengan *plugin* bernama Mason.

Untuk meningkatkan AstroNvim, kita dapat memasang *plugin* tambahan yang tersedia di internet. Ini akan membantu kita dalam meningkatkan fitur dan kemampuan Neovim dalam menunjang proses pengembangan. Dengan menggunakan AstroNvim sebagai *layer* IDE, kita dapat mempermudah dan meningkatkan proses pengembangan kita.

## Antarmuka

Sebelum mengoperasikan Neovim dengan konfigurasi AstroNvim, kita akan membahas hal yang paling mendasar terlebih dahulu, yaitu

antarmuka bagian awal ketika membuka Neovim dengan konfigurasi AstroNvim.



Q	Find File	SPC f f
█	Recents	SPC f o
█	Find Word	SPC f w
█	New File	SPC f n
█	Bookmarks	SPC f m

Tampilan awal Neovim

AstroNvim menggunakan *plugin* Alpha untuk menampilkan halaman *dashboard* atau *welcome screen* pada awal kita membuka Neovim. Tanpa *plugin* tersebut, umumnya Neovim hanya menampilkan tampilan awal bawaan Neovim saja.

Beralih pada tampilan yang lain, kita dapat membuat sebuah *buffer* baru di dalam Neovim dan akan memiliki tampilan seperti berikut:

A screenshot of a terminal window titled "Alacritty". The main area shows a buffer titled "init.lua" with the following Lua code:

```
1 local impatient_ok, impatient = pcall(require, "impatient")
1 if impatient_ok then impatient.enable_profile() end
2
3 for _, source in ipairs {
4     "core.utils",
5     "core.options",
6     "core.bootstrap",
7     "core.diagnostics",
8     "core.autocmds",
9     "core.mappings",
10    "configs.which-key-register",
11 } do
12    local status_ok, fault = pcall(require, source)
13    if not status_ok then vim.api.nvim_err_write("Failed to load " .. source .. "\n\n" .. fault) end
14 end
15
16 astronvim.conditional_func(astronvim.user_plugin_opts("polish", nil, false))
17
18 if vim.fn.has "nvim-0.8" ~= 1 or vim.version().prerelease then
19    vim.schedule(function() astronvim.notify("Unsupported Neovim Version! Please check the requirements", "error") end
20 end
```

### Tampilan pada saat membuka *buffer*

Pada bagian paling atas Neovim terdapat *bufferline*. AstroNvim menggunakan *plugin* Bufferline untuk menampilkan daftar *buffer* pada *bufferline*. Jika kita menampilkan banyak *buffer*, maka semuanya akan terlihat pada *bufferline* tersebut. Ini akan memudahkan kita dalam beralih dari satu *buffer* ke *buffer* yang lain.

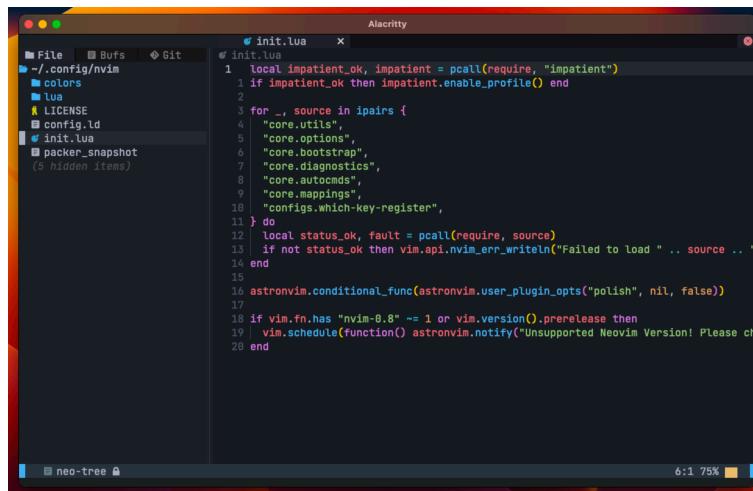
Di bagian paling bawah terdapat *statusline*, pada *statusline* terdapat banyak informasi mengenai mode yang sedang aktif, *branch* Git, informasi berkas, hingga informasi navigasi. AstroNvim menggunakan *plugin* Heirline untuk menampilkan *statusline* dengan cantik. Secara teknikal, masing-masing informasi tersebut diwakili oleh sebuah komponen, misal informasi Git dapat ditampilkan melalui komponen Git. Ini yang membuat Heirline menjadi *modular*.

Sayangnya, konfigurasi bawaan AstroNvim tidak menampilkan teks mode, sehingga akan kesulitan untuk mengetahui mode yang sedang aktif. Kita akan mengubah konfigurasi ini di bagian berikutnya.

## Panduan Dasar

Terdapat beberapa perintah dasar yang tersedia pada AstroNvim untuk kita menggunakan fitur-fitur dasar di dalanya, seperti membuka *file explorer*, beralih *buffer*, mencari berkas, dan masih banyak lagi.

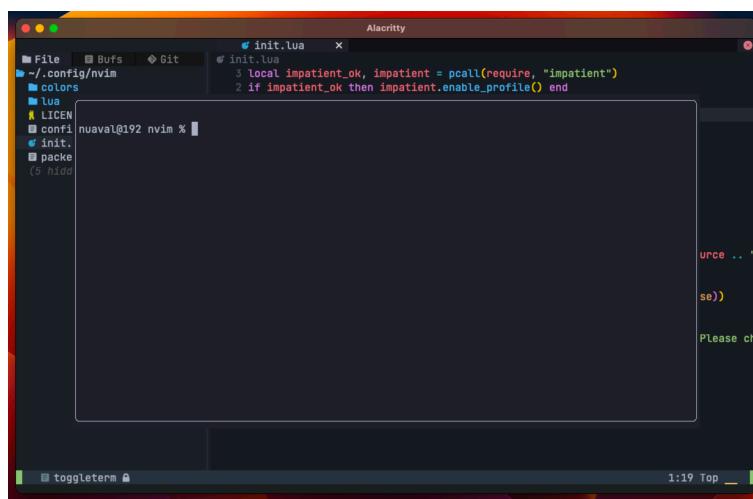
Kita dapat membuka *file explorer* dengan menggunakan perintah SPASI-e.



```
Alacrity
init.lua
1 local impatient_ok, impatient = pcall(require, "impatient")
2 if impatient_ok then impatient.enable_profile() end
3 for _, source in ipairs {
4     "core.utils",
5     "core.options",
6     "core.bootstrap",
7     "core.diagnostics",
8     "core.autocmds",
9     "core.mappings",
10    "configs.which-key-register",
11 } do
12     local status_ok, fault = pcall(require, source)
13     if not status_ok then vim.api.nvim_err_write("Failed to load " .. source .. " " .. fault)
14     end
15
16     astronvim.conditional_func(astronvim.user_plugin_opts("polish", nil, false))
17
18     if vim.fn.has "nvim-0.8" == 1 or vim.version().prerelease then
19         vim.schedule(function() astronvim.notify("Unsupported Neovim Version! Please ch
20         end
```

Membuka *file explorer*

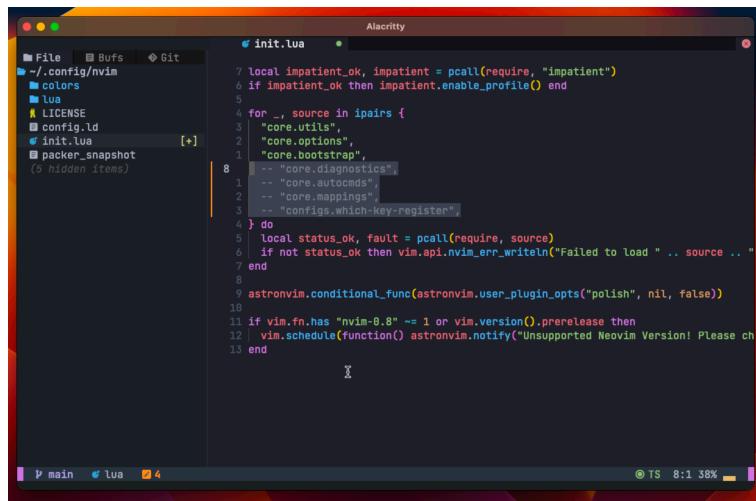
Kita juga dapat membuka terminal dengan menekan tombol F7.



```
Alacrity
init.lua
1 local impatient_ok, impatient = pcall(require, "impatient")
2 if impatient_ok then impatient.enable_profile() end
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

Membuka *terminal emulator* di dalam Neovim

Kita dapat mengomentari satu atau beberapa baris dengan menggunakan perintah SPASI-/-.



```
File   Bufs  Git
~/.config/nvim
  colors
  lua
  LICENSE
  config.ld
  init.lua  [+]
  packer_snapshot
(5 hidden items)

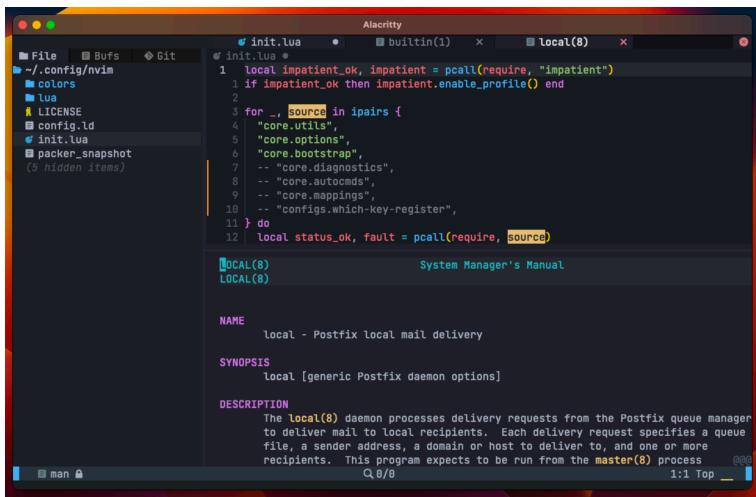
init.lua
1 local impatient_ok, impatient = pcall(require, "impatient")
2 if impatient_ok then impatient.enable_profile() end
3
4 for _, source in ipairs {
5   "core.utils",
6   "core.options",
7   "core.bootstrap",
8   -- "core.diagnostics",
9   -- "core.autocmds",
10  -- "core.mappings",
11  -- "configs.which-key-register",
12 } do
13   local status_ok, fault = pcall(require, source)
14   if not status_ok then vim.api.nvim_err_write("Failed to load " .. source .. " " .. fault)
15   end
16   astronvim.conditional_func(astronvim.user_plugin_opts("polish", nil, false))
17
18 if vim.fn.has "nvim-0.8" == 1 or vim.version().prerelease then
19   vim.schedule(function() astronvim.notify("Unsupported Neovim Version! Please ch
20   end

X

main  lua  4
TS 8:1 38%
```

Memberikan komentar pada beberapa baris kode sekaligus

Untuk melihat diagnosa baris, gunakan perintah `gl`. Kita dapat mengakses dokumentasi dengan perintah `K`. Untuk pergi ke definisi, gunakan perintah `gd`. Kita dapat menggunakan perintah `SPASI-1a` untuk membuka *code action*.



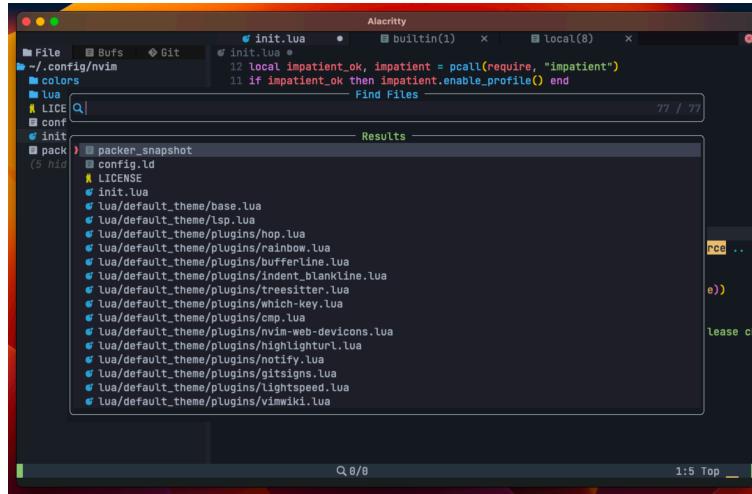
```
File   Bufs  Git
~/.config/nvim
  colors
  lua
  LICENSE
  config.ld
  init.lua
  packer_snapshot
(5 hidden items)

init.lua
1 local impatient_ok, impatient = pcall(require, "impatient")
2 if impatient_ok then impatient.enable_profile() end
3
4 for _, source in ipairs {
5   "core.utils",
6   "core.options",
7   "core.bootstrap",
8   -- "core.diagnostics",
9   -- "core.autocmds",
10  -- "core.mappings",
11  -- "configs.which-key-register",
12 } do
13   local status_ok, fault = pcall(require, source)
14
15 [MANUAL]                               System Manager's Manual
16 [LOCAL(8)]                            local - Postfix local mail delivery
17 [LOCAL(8)]
18
19 NAME          local - Postfix local mail delivery
20 SYNOPSIS      local [generic Postfix daemon options]
21 DESCRIPTION   The local(8) daemon processes delivery requests from the Postfix queue manager
22                  to deliver mail to local recipients. Each delivery request specifies a queue
23                  file, a sender address, a domain or host to deliver to, and one or more
24                  recipients. This program expects to be run from the master(8) process
25
26 Q 0/0  i:1 Top —
```

Mengakses dokumentasi dari bagian kode

Untuk menemukan berkas, gunakan perintah `SPASI-ff`. Untuk *grep* berkas, gunakan perintah `SPASI-fw`. Untuk mendapatkan status Git,

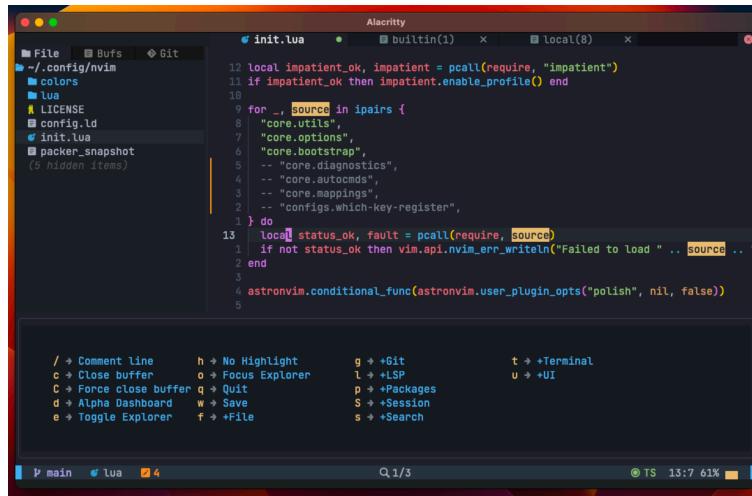
gunakan perintah **SPASI-gt**. Untuk menemukan berkas lama, gunakan perintah **SPASI-fo**.



The screenshot shows a terminal window titled "Alacritty". In the top right corner, there are three tabs: "init.lua", "builtin(1)", and "local(8)". The main pane displays a file search results for "init.lua". The results list includes files such as "LICENSE", "init.lua", "lua/default\_theme/base.lua", "lua/default\_theme/lsp.lua", "lua/default\_theme/plugins/hop.lua", "lua/default\_theme/plugins/rainbow.lua", "lua/default\_theme/plugins/bufferline.lua", "lua/default\_theme/plugins/indent\_blankline.lua", "lua/default\_theme/plugins/treesitter.lua", "lua/default\_theme/plugins/which-key.lua", "lua/default\_theme/plugins/cmp.lua", "lua/default\_theme/plugins/nvim-web-devicons.lua", "lua/default\_theme/plugins/highlighturl.lua", "lua/default\_theme/plugins/notify.lua", "lua/default\_theme/plugins/gitsigns.lua", "lua/default\_theme/plugins/lightspeed.lua", and "lua/default\_theme/plugins/vimwiki.lua". A status bar at the bottom indicates "Q 0/0" and "1:5 Top".

Menemukan berkas

Kita dapat menggunakan *plugin which key* untuk mendapatkan *menu* dari beberapa *binding* perintah yang berguna dengan menekan tombol **SPASI**.



The screenshot shows a terminal window titled "Alacritty". In the top right corner, there are three tabs: "init.lua", "builtin(1)", and "local(8)". The main pane displays the content of the "init.lua" file. Below the code, a list of keybindings is shown:

/	→ Comment line	h	→ No Highlight	g	→ +git	t	→ +Terminal
c	→ Close buffer	o	→ Focus Explorer	l	→ +LSP	u	→ +UI
C	→ Force close buffer	q	→ Quit	p	→ +Packages		
d	→ Alpha Dashboard	w	→ Save	S	→ +Session		
e	→ Toggle Explorer	f	→ +File	s	→ +Search		

A status bar at the bottom indicates "Q 1/3", "TS 13:7 61%", and battery level.

Menampilkan *keybinding*

Kita dapat menavigasi *buffer* dengan perintah **H** untuk beralih ke *buffer* kiri atau **L** untuk beralih ke *buffer* kanan. Untuk menavigasi *window*,

gunakan `CTRL-h` untuk beralih ke *window* kiri, `CTRL-l` untuk beralih ke *window* kanan, `CTRL-k` untuk beralih ke *window* atas, dan `CTRL-j` untuk beralih ke *window* bawah. Untuk menutup *buffer* saat ini, gunakan perintah `SPASI-c`.

Kita dapat mengubah ukuran *buffer* dengan menekan `CTRL` ditambah tombol panah ke arah yang diinginkan (kiri, kanan, atas, bawah).

Selain perintah-perintah di atas, AstroNvim dibekali dengan pemetaan bawaan atau *default mapping* yang dapat kita gunakan untuk melakukan aksi-aksi tertentu. Pemetaan tersebut dibagi menjadi beberapa kategori, di antaranya:

## Umum

Berikut adalah pemetaan untuk kategori umum yang di mana pemetaan atau perintah-perintah tersebut yang biasa digunakan hal-hal yang umum dilakukan, seperti kembali ke mode normal, beralih *buffer*, memberi komentar dan yang lainnya:

- Tombol *leader*: `SPASI`
- Tombol *escape*: `jj`, `jk`
- Memperbesar *window*: `CTRL-Atas`
- Memperkecil *window*: `CTRL-Bawah`
- Memperkecil *window* ke kiri: `CTRL-Kiri`
- Memperbesar *window* ke kanan: `CTRL-Kanan`
- Memindahkan kursor ke *window* atas: `CTRL-k`
- Memindahkan kursor ke *window* bawah: `CTRL-j`
- Memindahkan kursor ke *window* kiri: `CTRL-h`
- Memindahkan kursor ke *window* kanan: `CTRL-l`
- Menyimpan secara paksa: `CTRL-s`
- Keluar secara paksa: `CTRL-q`
- Membuat berkas baru: `SPASI-fn`
- Menutup *buffer*: `SPASI-c`
- Buffer selanjutnya: `SHIFT-1`

- Buffer sebelumnya: SHIFT-h
- Komentar: SPASI- /

Tombol *leader* merupakan tombol yang digunakan sebagai tombol kombinasi dengan tombol lainnya untuk mengakses fitur-fitur yang tersedia, seperti perintah SPASI-e untuk membuka *file explorer*, di mana SPASI merupakan tombol *leader* sebagai tombol kombinasi tombol e.

## Bufferline

Perintah-perintah berikut digunakan untuk mengakses fitur yang terdapat di *plugin Bufferline*, di antaranya:

- Memindahkan *buffer* ke kanan: >b
- Memindahkan *buffer* ke kiri: <b

## Neo-Tree

Perintah-perintah berikut digunakan untuk mengakses fitur yang terdapat di *plugin Neo-Tree*, di antaranya:

- Membuka dan menutup: SPASI-e
- Fokus: SPASI-o

## Dashboard

Perintah berikut digunakan untuk mengakses tampilan *dashboard*:

- Pergi ke *dashboard*: SPASI-d

## Session Manager

Perintah-perintah berikut digunakan untuk mengelola sesi di dalam Neovim, di antaranya:

- Simpan Sesi: SPASI-Ss
- Sesi Terakhir: SPASI-S1

- Hapus Sesi: SPASI-Sd
- Cari Sesi: SPASI-Sf
- Muat Direktori Saat Ini Sesi: SPASI-S.

## Package Management

Perintah-perintah berikut digunakan untuk mengelola paket di dalam Neovim yang di mana menggunakan *plugin* Packer, di antaranya:

- Pembaruan paket AstroNvim: SPASI-pa
- Pembaruan AstroNvim: SPASI-pA
- Versi AstroNvim: SPASI-pv
- Pemasang Mason: SPASI-pI
- Pembaru Mason: SPASI-pU
- Kompilasi Packer: SPASI-pc
- Memasang Packer: SPASI-pi
- Status Packer: SPASI-ps
- Sinkronisasi Packer: SPASI-ps
- Pembaruan Packer: SPASI-pu

## Language Server Protocol

Perintah-perintah berikut digunakan untuk mengelola LSP di dalam Neovim, di antaranya:

- Informasi LSP: SPASI-li
- Dokumen *hover*: SHIFT-k
- Format dokumen: SPASI-lf
- Outline simbol: SPASI-lS
- Diagnostik baris: gl, SPASI-ld
- Semua diagnostik: SPASI-1D
- Tindakan kode: SPASI-1a
- Bantuan tanda tangan: SPASI-1h
- Ganti nama: SPASI-1r
- Simbol dokumen: SPASI-1s

- Simbol lingkup kerja: SPASI-1G
- Diagnostik berikutnya: ]d
- Diagnostik sebelumnya: [d
- Deklarasi: gD
- Definisi Tipe: gT
- Definisi: gd
- Implementasi: gI
- Referensi: gr

## Telescope

Perintah-perintah berikut digunakan untuk mengakses fitur Telescope di dalam Neovim, di antaranya:

- *Live grep*: SPASI-fw
- *Live grep* (termasuk yang tersembunyi): SPASI-fW
- Status Git: SPASI-gt
- Cabang Git: SPASI-gb, SPASI-sb
- *Commit* Git: SPASI-gc
- Mencari berkas: SPASI-ff
- Mencari berkas (termasuk yang tersembunyi): SPASI-ff
- *Buffer*: SPASI-fb
- *Tag* bantuan: SPASI-fh, SPASI-sh
- Tanda: SPASI-fm
- Berkas lama: SPASI-fo
- Halaman *man*: SPASI-sm
- Notifikasi: SPASI-sn
- Daftar: SPASI-sr
- Peta kunci: SPASI-sk
- Perintah: SPASI-sc
- Simbol LSP: SPASI-ls
- Simbol lingkup kerja LSP: SPASI-1G
- Referensi LSP: SPASI-1R
- Diagnostik LSP: SPASI-1D

## Toggle Terminal

Perintah-perintah berikut digunakan untuk mengakses fitur *terminal* di dalam Neovim, di antaranya:

- Beralih *terminal*: F7
- *Terminal* melayang: SPASI-tf
- Membagi *terminal* horizontal: SPASI-th
- Membagi *terminal* vertikal: SPASI-tv
- LazyGit: SPASI-tl, SPASI-gg
- Node: SPASI-tn
- GDU: SPASI-tu
- Bottom: SPASI-tt

## Git

Perintah-perintah berikut digunakan untuk mengakses fitur Git di dalam Neovim, di antaranya:

- *Hunk* berikutnya: SPASI-gj
- *Hunk* sebelumnya: SPASI-gk
- *Blame* baris: SPASI-g1
- Pratinjau *hunk*: SPASI-gp
- Atur ulang *hunk*: SPASI-gr
- Tingkatkan *hunk*: SPASI-gs
- *Hunk* tanpa tingkat: SPASI-gu
- Git *diff*: SPASI-gd

## User Interface

Perintah-perintah berikut digunakan untuk mengakses fitur-fitur yang berkaitan dengan antarmuka pengguna di dalam Neovim, di antaranya:

- Menghidupkan/Mematikan *autopairs*: SPASI-ua
- Menghidupkan/Mematikan latar belakang: SPASI-ub

- Menghidupkan/Mematikan *completion*: SPASI-uc
- Menghidupkan/Mematikan penyorotan warna: SPASI-uC
- Menghidupkan/Mematikan diagnostik: SPASI-ud
- Menghidupkan/Mematikan pemformatan otomatis: SPASI-uf
- Menghidupkan/Mematikan *signcolumn*: SPASI-ug
- Mengubah pengaturan indentasi: SPASI-ui
- Menghidupkan/Mematikan *statusline*: SPASI-ul
- Mengubah penomoran baris: SPASI-un
- Menghidupkan/Mematikan mode *paste*: SPASI-up
- Menghidupkan/Mematikan pemeriksaan ejaan: SPASI-us
- Menghidupkan/Mematikan *conceal*: SPASI-uS
- Menghidupkan/Mematikan *tabline*: SPASI-ut
- Menghidupkan/Mematikan penyorotan URL: SPASI-uu
- Menghidupkan/Mematikan pembungkus: SPASI-uw
- Menghidupkan/Mematikan penyorotan sintaks: SPASI-uy

## Konfigurasi

Pada dasarnya AstroNvim hanya sekadar sekumpulan konfigurasi yang memudahkan kita dalam menjadikan Neovim sebagai IDE, bukan sebuah proyek *fork* Vim layaknya Neovim. Sebelumnya kita sudah belajar mengenai penggunaan dasar fitur-fitur yang terdapat pada AstroNvim, fitur-fitur tersebut tersedia berkat adanya konfigurasi dari AstroNvim. Kendati hal ini membuat Neovim kita menjadi *opinionated*, tapi sebenarnya kendali konfigurasi tetap berada di tangan kita.

Konfigurasi AstroNvim dapat kita kustomisasi sesuai dengan keinginan dan kebutuhan, mungkin saja kita ingin mengubah, menambah, atau bahkan menghapus suatu konfigurasi yang terdapat pada AstroNvim secara bawaan. Hal ini dapat dilakukan dengan mudah, karena pada dasarnya sama seperti konfigurasi Neovim pada umumnya, hanya saja ditulis oleh AstroNvim.

Ketika Neovim dijalankan, ia akan membaca berkas konfigurasi yang kita tulis di dalam alamat sesuai dengan sistem operasi yang kita

gunakan, seperti `~/.config/nvim/init.vim` (atau `.lua`) untuk pengguna Unix dan Unix-like atau `~/AppData/Local/nvim/init.vim` (atau `.lua`) untuk pengguna Windows.

Karena kita menggunakan konfigurasi AstroNvim, maka Neovim akan membaca konfigurasi tersebut, konfigurasi yang ditulis oleh AstroNvim. Kita dapat lihat berkas `init.lua` dari AstroNvim seperti ini:

```
local impatient_ok, impatient = pcall(require, "impatient")

if impatient_ok then impatient.enable_profile() end

for _, source in ipairs {

    "core.utils",
    "core.options",
    "core.bootstrap",
    "core.diagnostics",
    "core.autocmds",
    "core.mappings",
    "configs.which-key-register",
} do

    local status_ok, fault = pcall(require, source)

    if not status_ok then
        vim.api.nvim_err_write("Failed to load " .. source
        .. "\n\n" .. fault) end

end

astronvim.conditional_func(as-
tronvim.user_plugin_opts("polish", nil, false))
```

```
if vim.fn.has "nvim-0.8" ~= 1 or vim.version().prerelease then  
    vim.schedule(function() astronvim.notify("Unsupported Neovim Version! Please check the requirements", "error") end)  
end
```

Tentu saja konfigurasi tersebut terlihat lebih rumit dari konfigurasi yang kita pelajari sebelumnya pada bagian konfigurasi Neovim. Hal ini bisa terjadi karena terdapat banyak sekali konfigurasi AstroNvim yang mengatur segala aspek dari Neovim, mulai dari pemetaan tombol, pengelolaan paket, pengaturan opsi, hingga memungkinkan kita dapat menulis konfigurasi sendiri tanpa menganggu konfigurasi bawaan AstroNvim.

Pada dasarnya, kita bisa saja mengubah konfigurasi bawaan AstroNvim agar sesuai dengan kehendak kita dengan mengubah berkas *init.lua* tersebut. Namun, itu bukan cara yang benar menurut AstroNvim, karena ini akan membuat konfigurasi AstroNvim tercampur dengan konfigurasi yang kita tulis sehingga akan membuat AstroNvim sulit atau bahkan tidak bisa untuk diperbarui.

## Kustomisasi

Kita dapat menulis konfigurasi sendiri untuk mengkustomisasi konfigurasi AstroNvim dengan cara membuat berkas konfigurasi baru di dalam folder *user*, lengkapnya seperti ini untuk Unix dan *Unix-like*:

```
~/config/nvim/lua/user/init.lua
```

Untuk Windows:

```
~/AppData/Local/nvim/lua/user/init.lua
```

Konfigurasi tersebut dapat ditulis dengan sintaksis seperti ini:

```
local config = {
```

```
}

return config
```

Seluruh konfigurasi personal yang kita inginkan dapat diletakkan di dalam *table config*. Sebagai contoh yang paling dasar, saya ingin mengubah tinggi *command-line* menjadi 1:

```
local config = {

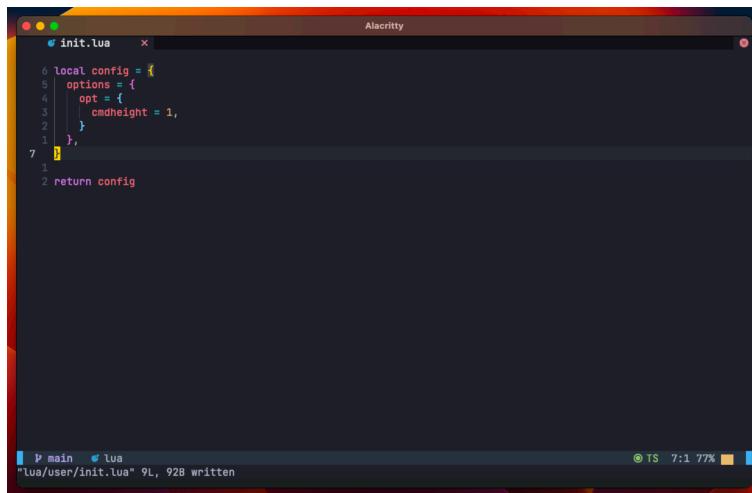
    options = {

        opt = {

            cmdheight = 1,
        }
    }
}

return config
```

Konfigurasi yang sebelumnya kita tulis dengan sintaksis `vim.opt.nama_opsi`, sekarang kita dapat tulis di dalam `options.opt.nama_opsi` seperti di atas. Karena dengan seperti itu AstroNvim dapat membaca konfigurasi personal yang kita tulis. Konfigurasi tersebut akan membuat *statusline* tidak lagi tergabung dengan baris *command-line*.



```
Alacritty
● init.lua x
6 local config = {
5   options = {
4     opt = {
3       cmdheight = 1,
2     }
1   },
7   }
1
2 return config

P main w lua
"lua/user/init.lua" 9L, 928 written
◎ TS 7:1 77%
```

Konfigurasi cmdheight = 1

Sama halnya dengan opsi-opsi Neovim yang lain, kita dapat menulis dengan cara serupa seperti sebelumnya.

## Dashboard

Bila kamu membuka Neovim tanpa argumen nama berkas apapun, maka Neovim akan membuka halaman *dashboard* atau kamu dapat menyebutnya *welcome screen* jika mau.



Neovim *dashboard*

Tulisan besar "ASTRO NVIM" yang kita lihat berasal dari konfigurasi bawaan AstroNvim, kita dapat mengubahnya pada konfigurasi header, misal seperti ini:

```
local config = {  
  
    header = {  
  
        "Nauval",  
  
        "@mhdnauvalazhar",  
  
    }  
  
    return config
```

Hasilnya akan seperti ini:



Konfigurasi *header*

Tentu saja tidak terlihat seperti sebelumnya yang dibuat oleh AstroNvim, untuk membuatnya nampak lebih bagus, kita dapat menggunakan alat seperti <https://fsymbols.com/generators/carty/>.

Nauval

X Copy



Copy



Copy

<https://fsymbols.com/generators/carty/>

Untuk menggunakannya, salin pada bagian yang diinginkan, kemudian taruh pada konfigurasi header seperti sebelumnya, dengan penulisan seperti ini:

```
1 local config = {
2   header = {
3     text = "NAUVAL"
4   },
5   options = {
6     opt = {
7       cmdheight = 1,
8     }
9   }
10 }
11 }
12 }
13 }
14 }
15 return config
```

lua/main.lua 17L, 1076B written

Konfigurasi *header*

Untuk melihat hasilnya, kita dapat menjalankan ulang Neovim.

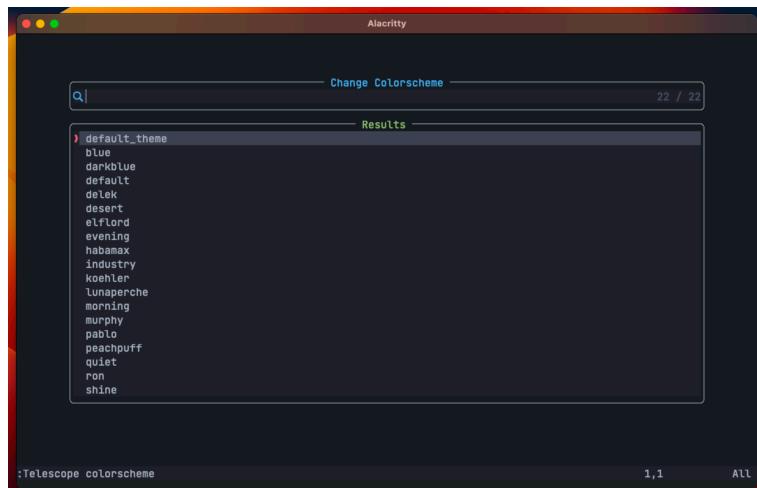


Neovim dengan *header* baru

Tentu hasilnya lebih bagus ketimbang sebelumnya yang hanya menggunakan teks reguler.

## Colorscheme

Secara bawaan konfigurasi AstroNvim menggunakan *default\_theme* sebagai *colorscheme*. Kita dapat mengubahnya sesuai yang kita inginkan. Sebelum itu, periksa nama *colorscheme* yang hendak digunakan dengan menggunakan perintah :Telescope colorscheme di dalam Neovim.



Daftar *colorscheme*

Setelah mendapat nama yang diinginkan, kita dapat mengaturnya dengan konfigurasi berikut:

```
local config = {  
    colorscheme = "blue",  
}  
  
return config
```

Seandainya kita hendak menggunakan *colorscheme* pihak ketiga yang lain, misal *colorscheme* bernama Tokyonight, kita dapat menggunakan sintaksis seperti berikut:

```
local config = {  
    colorscheme = "tokyonight",  
    plugins = {  
        init = {  
            {"folke/tokyonight.nvim",  
        },  
    },  
}
```

```
    as = "tokyonight",
    config = function()
        require("tokyonight").setup {}
    end,
},
},
},
}

return config
```

Konfigurasi tersebut akan menggunakan *colorscheme* Tokyonight dan memasang *plugin* tersebut karena Tokyonight merupakan *colorscheme* pihak ketiga.

Jalankan ulang *program* Neovim, kemudian jalankan perintah :Pack-erSync di dalam Neovim untuk memasang *plugin* Tokyonight yang sebelumnya kita tulis di dalam konfigurasi. Tunggu hingga prosesnya selesai dan jalankan ulang Neovim kembali untuk membuat Neovim menggunakan *colorscheme* yang baru.



Neovim dengan Tokyonight

Pada *colorscheme* Tokyonight terdapat beberapa gaya, seperti *storm*, *moon*, *night*, dan *day*. Gaya *storm* digunakan sebagai bawaan. Anggap saja kita ingin menggunakan gaya *night*, untuk memberikan konfigurasinya, kita dapat menggunakan cara seperti ini:

```
local config = {

    colorscheme = "tokyonight",

    plugins = {

        init = {

            "folke/tokyonight.nvim",
            as = "tokyonight",
            config = function()

                require("tokyonight").setup({
                    style = "night"
                })
            end
        }
    }
}
```

```

        end,
    },
},
},
}

return config

```

Jalankan ulang Neovim dan jalankan perintah :PackerSync kembali, tunggu hingga prosesnya selesai dan jalankan Neovim kembali untuk memulai menggunakan *colorscheme* dengan gaya yang baru.



Tokyonight dengan gaya *night*

Gaya *night* pada Tokyonight memiliki tema yang lebih gelap dibanding sebelumnya. Semua konfigurasi yang berkaitan dengan *colorscheme* dapat dituliskan dengan cara seperti sebelumnya.

## Statusline

AstroNvim menggunakan *plugin* Heirline untuk mengatur antarmuka *statusline*. Pada Neovim, *statusline* berada di bagian bawah dan di atas

*command-line*. Seperti yang kita lihat saat ini *statusline* tidak menampilkan teks mode yang sedang aktif saat ini, seperti normal, *insert*, *visual*, atau mode yang lain.

Menampilkan teks mode pada *statusline* tentu sangat berguna bagi pemula yang masih baru menggunakan Neovim. Untuk itu kita akan mengubah konfigurasi *statusline* bawaan AstroNvim agar menampilkan teks mode.

*Statusline* terdiri dari beberapa komponen di dalamnya, seperti *mode*, *branch* Git, informasi berkas, *diagnostic* hingga *nav*. Secara bawaan, komponen mode terdapat pada statusline, hanya saja tidak menampilkan teks, maka dari itu kita perlu mengubah komponen moder tersebut. Untuk itu, kita dapat menggunakan konfigurasi berikut:

```
local config = {

    plugins = {

        heirline = function(config)

            config[1] = {

                hl = { fg = "fg", bg = "bg" },

                astronvim.status.component.mode { mode_text =
{ padding = { left = 1, right = 1 } } },

                astronvim.status.component.git_branch(),
                astronvim.status.component.file_info(),
                astronvim.status.component.git_diff(),
                astronvim.status.component.diagnostics(),
                astronvim.status.component.fill(),
                astronvim.status.component.macro_recording(),
                astronvim.status.component.fill(),
                astronvim.status.component.lsp(),
            }
        end
    }
}
```

```

astronvim.status.component.treesitter(),
astronvim.status.component.nav(),
}
end,
},
return config

```

Pada konfigurasi di atas bagian yang penting untuk menampilkan teks mode di *statusline* adalah:

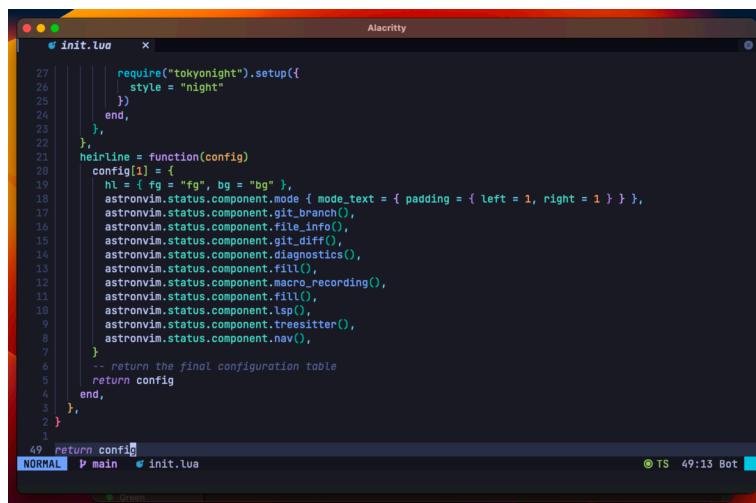
```

astronvim.status.component.mode { mode_text = { padding = { left = 1, right = 1 } } },

```

Hanya saja, agar komponen yang lain juga ditampilkan kembali, maka kita perlu menulis ulang atau menyertakan komponen-komponen tersebut kembali.

Jalankan ulang Neovim untuk melihat hasilnya.



Neovim dengan teks mode

Kini *statusline* memiliki teks mode yang akan memudahkan kita mengetahui mode yang sedang aktif saat ini.

## Plugin

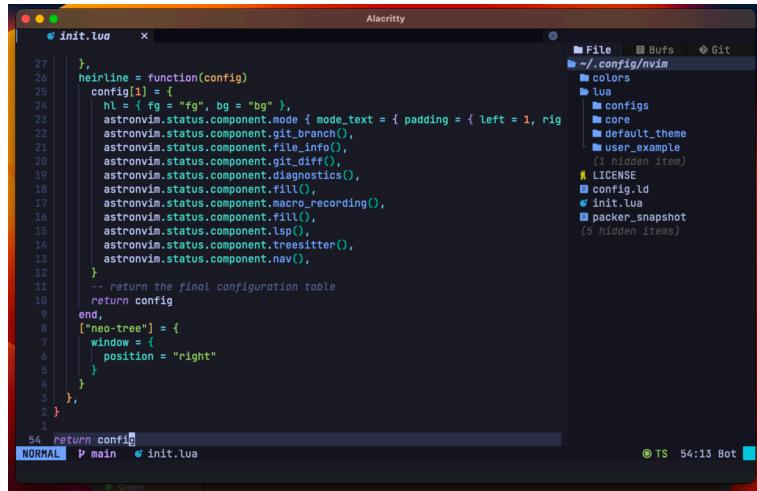
Seperti yang sudah dijelaskan sebelumnya bahwa konfigurasi AstroNvim bergantung pada beberapa *plugin-plugin* pihak ketiga. Sebagai contoh, sebelumnya *statusline* bergantung pada *plugin* Heirline.

Seluruh *plugin-plugin* tersebut memiliki konfigurasi atau opsi-opsi bawaan yang sudah diatur oleh AstroNvim. Kendati sudah diatur secara bawaan, kita juga dapat mengubah konfigurasi pada *plugin-plugin* tertentu sesuai dengan keinginan kita, seperti yang kita lakukan pada *statusline* sebelumnya.

Untuk mengubah konfigurasi pada *plugin* tertentu di AstroNvim, kita dapat melakukannya dengan cara yang sama seperti kita mengubah konfigurasi pada *plugin* Heirline. Sebagai contoh, kita akan mengubah konfigurasi pada *plugin* Neo-tree agar posisinya berpindah ke sebelah kanan Neovim. Untuk melakukannya, kita dapat menggunakan cara seperti berikut:

```
local config = {  
    plugins = {  
        ["neo-tree"] = {  
            window = {  
                position = "right"  
            }  
        }  
    }  
    return config
```

Jalankan kembali Neovim untuk melihat hasilnya.



The screenshot shows a terminal window titled "Alacritty" with the file "init.lua" open. The code in the terminal is as follows:

```
27 },
28   heirline = function(config)
29     config[1] = {
30       hl = { fg = "#e9", bg = "#00" },
31       astronvim.status.component.mode { mode_text = { padding = { left = 1, rig
32       astronvim.status.component.git_branch(),
33       astronvim.status.component.file_info(),
34       astronvim.status.component.git_diff(),
35       astronvim.status.component.diagnostics(),
36       astronvim.status.component.fill(),
37       astronvim.status.component.macros_recording(),
38       astronvim.status.component.fill(),
39       astronvim.status.component.lsp(),
40       astronvim.status.component.treesitter(),
41       astronvim.status.component.nav(),
42     }
43   }
44   -- return the final configuration table
45   return config
46 end,
47 ["neo-tree"] = {
48   window = {
49     position = "right"
50   }
51 },
52 }
53
54 return config
```

Below the terminal, the status bar shows "NORMAL" and "main". To the right of the terminal is a file browser window titled "File" showing the directory structure of the nvim configuration folder. The "init.lua" file is highlighted in the list.

Posisi Neo-tree di sebelah kanan

Begitu juga dengan *plugin-plugin* yang lain, kita dapat menggunakan cara yang sama untuk mengubah konfigurasi bawaannya – semuanya ditaruh di dalam *key plugins*.

Selain itu, kita dapat menonaktifkan *plugin* bawaan konfigurasi AstroNvim. Sebagai contoh, kita akan menonaktifkan *plugin* Alpha:

```
local config = {

  plugins = {

    init = {

      ["goolord/alpha-nvim"] = { disable = true },

    }

  }

  return config
}
```

Jalankan ulang Neovim, lalu jalankan perintah :PackerSync agar Packer melakukan *cleanup* terhadap *plugin* yang tidak digunakan.

Kamu akan mendapat *floating window* untuk mengkonfirmasi penghapusan *plugin* dari Packer, masukkan *y* dan tekan ENTER.

Ketika kita memuat ulang Neovim, maka sudah tidak ada lagi *dashboard* atau *welcome screen* karena kita sudah menonaktifkan *plugin* tersebut. Untuk mengembalikannya, hapus saja konfigurasi yang sebelumnya kita tulis untuk menonaktifkan *plugin* tersebut, dan sisanya lakukan langkah yang sama seperti sebelumnya.

Selain itu, kita dapat menambah *plugin* baru yang hendak kita pasang ke dalam Neovim. Misal, kita hendak memasang *plugin* Emmet di Neovim, kita dapat menggunakan konfigurasi berikut:

```
local config = {  
    plugins = {  
        init = {  
            { "mattn/emmet-vim" }  
        }  
    }  
}  
  
return config
```

Jalankan ulang Neovim dan jalankan perintah `:PackerSync` kembali, tunggu hingga prosesnya selesai dan jalankan Neovim kembali agar memuat *plugin* yang baru.

Untuk mengujinya, kita dapat buat berkas HTML baru dan ketik kode berikut:

```
ul>li>a*5
```

Lalu gunakan kombinasi tombol `CTRL-y`, untuk mengubahnya menjadi kode HTML seperti berikut:

```
<ul>
```

```
<li>  
    <a href=""></a>  
    <a href=""></a>  
    <a href=""></a>  
    <a href=""></a>  
    <a href=""></a>  
</li>  
</ul>
```

Untuk memasang *plugin-plugin* yang lain kita dapat menggunakan cara yang sama.

# Syntax Highlighter & Language Server Protocol

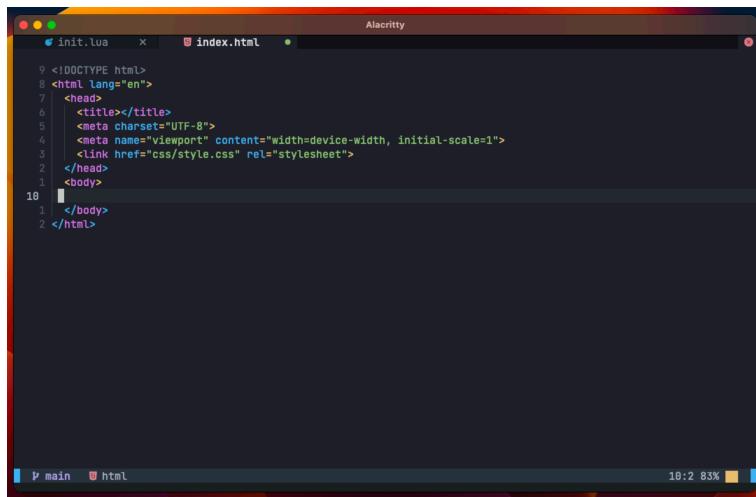
Pada bagian ini kita akan membahas prihal *syntax highlighter* dan Language Server Protocol (LSP). Kedua hal ini didukung secara bawaan oleh Neovim, namun kita perlu mempelajari cara mengelola dan menggunakananya.

## Syntax Highlighter

Secara bawaan, Neovim memiliki *syntax highlighter* untuk memberikan warna pada setiap elemen-elemen kode yang kita tulis di dalamnya. Namun, dalam beberapa kasus, perwarnaan yang diberikan seringkali tidak begitu baik, sehingga elemen-elemen memiliki warna yang sama dengan elemen jenis lain.

AstroNvim menggunakan *plugin* Treesitter untuk memberikan pewarnaan pada sintaksis dengan lebih cantik dibandingkan dengan hasil pada pewarnaan tradisional.

Anggap saja kita memiliki kode HTML seperti berikut:

A screenshot of the Alacritty terminal window. It shows two tabs: 'init.lua' and 'index.html'. The 'index.html' tab is active and displays the following HTML code:

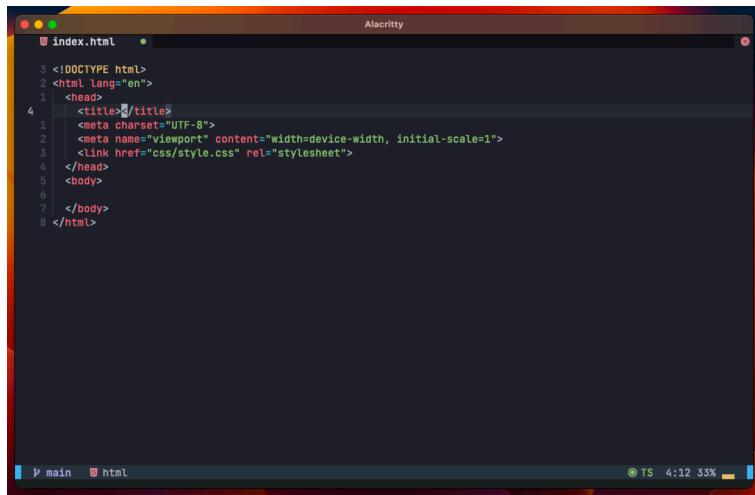
```
9 <!DOCTYPE html>
8 <html lang="en">
7 <head>
6   <title></title>
5   <meta charset="UTF-8">
4   <meta name="viewport" content="width=device-width, initial-scale=1">
3   <link href="css/style.css" rel="stylesheet">
2 </head>
1 <body>
10
1 </body>
2 </html>
```

The code is color-coded: blue for tags like <html>, <head>, <title>, <meta>, <link>, <body>, and </>; red for attributes like lang="en", charset="UTF-8", name="viewport", content="width=device-width, initial-scale=1", href="css/style.css", and rel="stylesheet"; and green for the numbers 9, 8, 7, 6, 5, 4, 3, 2, 1, and 10.

*Syntax highlighter* pada kode HTML

Kita dapat memasang *syntax highlighter* pada Treestitter untuk masing-masing jenis sintaksis. Sebagai contoh untuk sintaksis HTML, kita dapat memasangnya dengan perintah :TSInstall html. Tunggu hingga proses pemasangan selesai dan jalankan ulang Neovim.

Hasilnya akan seperti ini:

A screenshot of the Alacritty terminal window titled "Alacritty". The window contains the following HTML code:

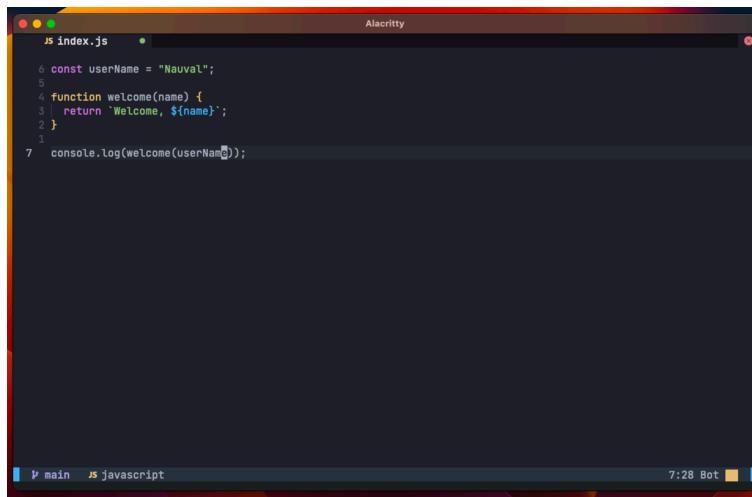
```
index.html
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <title>/title</title>
5     <meta charset="UTF-8">
6     <meta name="viewport" content="width=device-width, initial-scale=1">
7     <link href="css/style.css" rel="stylesheet">
8   </head>
9   <body>
10  </body>
11 </html>
```

The code is color-coded: blue for tags, red for attributes, and green for values. The status bar at the bottom shows "TS 4:12 33%".

*Syntax highlighter* dengan Treesitter

Tentu hasil yang sekarang lebih baik dibanding yang sebelumnya. Untuk sintaksis yang lain kita dapat memasangnya dengan cara yang sama seperti sebelumnya. Sebagai contoh, kita akan memasang untuk sintaksis JavaScript.

Sebelumnya seperti ini:



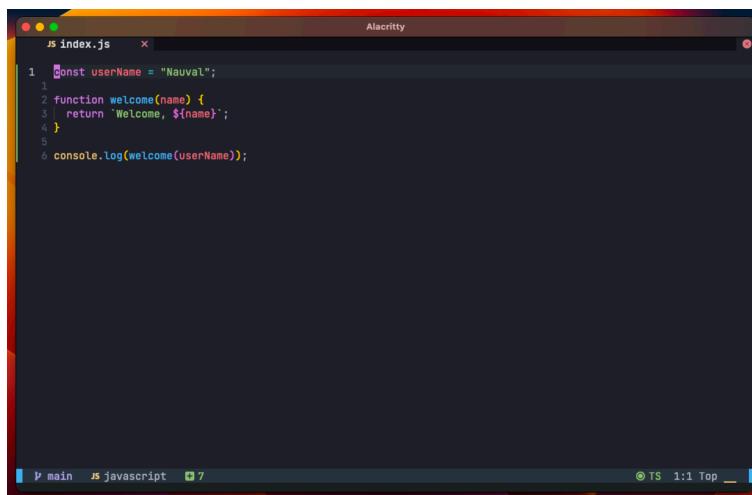
A screenshot of a terminal window titled "Alacritty". The window has a dark theme with orange window controls. Inside, there is a single tab labeled "index.js" containing the following JavaScript code:

```
1 const userName = "Nauval";
2 function welcome(name) {
3 | return `Welcome, ${name}`;
4 }
5
6 console.log(welcome(userName));
```

The terminal status bar at the bottom shows "main" and "javascript" under the "File" tab, and the time "7:28 Bot".

Kode JavaScript tanpa Treesitter

Setelah memasang sintaksis untuk JavaScript, hasilnya seperti ini:



A screenshot of a terminal window titled "Alacritty". The window has a dark theme with orange window controls. Inside, there is a single tab labeled "index.js" containing the same JavaScript code as the previous screenshot, but with syntax highlighting. The word "const" is blue, "userName" is green, "function" is purple, "welcome" is red, and "console" is light blue. The terminal status bar at the bottom shows "main" and "javascript" under the "File" tab, and the time "7:28 Bot". Additionally, there is a status bar at the bottom right showing "TS 1:1 Top".

Kode JavaScript dengan Treesitter

Kamu dapat memasang jenis sintaksis yang pada Treesitter dengan cara yang sama seperti sebelumnya sesuai dengan kebutuhanmu.

## Language Server Protocol

Language Server Protocol (LSP) adalah suatu spesifikasi yang menentukan bagaimana suatu *server* yang menyediakan layanan untuk menyunting dan menganalisis kode dari suatu bahasa pemrograman tertentu dapat berkomunikasi dengan IDE yang menggunakannya.

LSP memungkinkan IDE untuk memanfaatkan layanan yang diberikan oleh *server* tersebut, seperti *autocomplete*, *go-to-definition*, *documentation*, *diagnostic*, *format* atau yang lainnya, dengan cara yang terstandarisasi. Dengan menggunakan LSP, pengembang dapat membuat IDE yang lebih kuat dan efisien, serta memudahkan pengembangan yang ditulis dalam bahasa pemrograman tertentu.

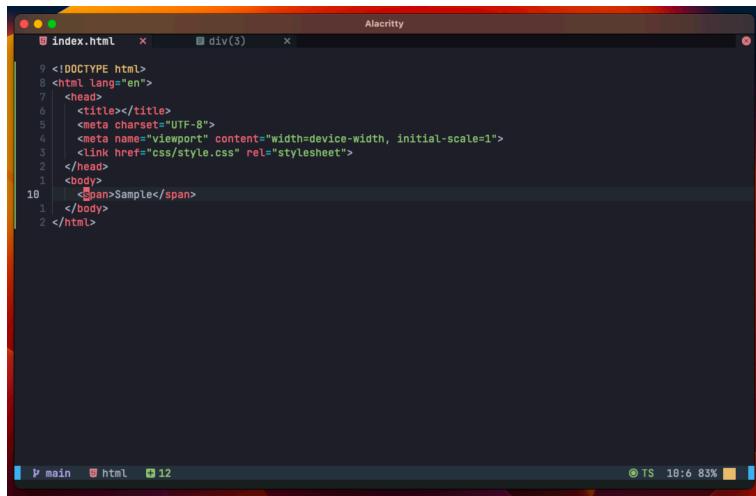
Perlu dicatat bahwa tidak semua *server* LSP menyediakan kapabilitas yang sama, sebagai contoh mungkin *server* LSP untuk HTML mendukung *code formatting* tapi tidak untuk *server* LSP CSS, begitu juga dengan fitur-fitur LSP yang lainnya. Ketersediaan fitur-fitur tersebut tergantung pada *server* LSP.

Neovim mendukung LSP secara bawaan, ini memungkinkan kita untuk memanfaatkan layanan yang diberikan oleh sebuah *server* LSP untuk menyunting dan menganalisis kode dari bahasa pemrograman tertentu.

Kendati begitu, pada dasarnya kita perlu mengkonfigurasi Neovim agar dapat bekerja dengan LSP dan memasang *server* LSP yang sesuai dengan bahasa pemrograman yang ingin kita gunakan. Namun, karena kita sudah menggunakan konfigurasi AstroNvim, kita tidak perlu melakukan konfigurasi lagi dari awal, seperti memasang paket *nvim-lspconfig*, *nvim-cmp*, hingga *mason*.

Untuk mulai menggunakan fitur-fitur LSP seperti *autocomplete* pada Neovim dengan konfigurasi AstroNvim kita hanya perlu memasang masing-masing *server* LSP untuk setiap pemrograman yang kita gunakan.

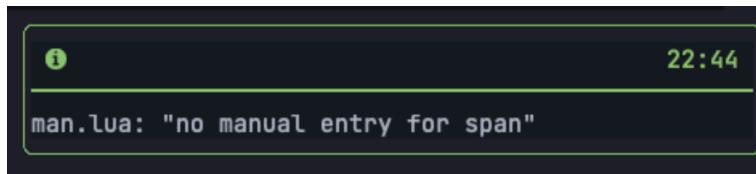
Sebagai contoh, kita dapat membuat berkas HTML baru seperti berikut:



```
index.html      div(3)      Alacritty
9 <!DOCTYPE html>
8 <html lang="en">
7   <head>
6     <title></title>
5     <meta charset="UTF-8">
4     <meta name="viewport" content="width=device-width, initial-scale=1">
3     <link href="css/style.css" rel="stylesheet">
2   </head>
1   <body>
10     <span>Sample</span>
1   </body>
2 </html>
```

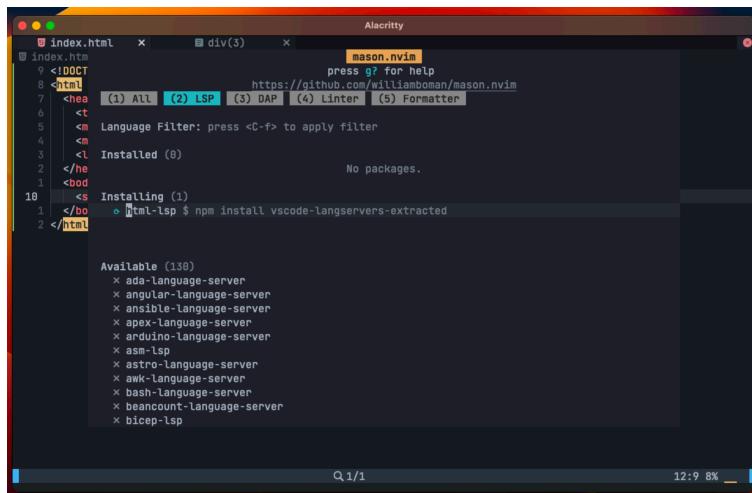
Berkas baru HTML

Jika kamu letakkan kursor di atas *tag span* lalu jalankan perintah **K**, maka akan muncul notifikasi di dalam Neovim seperti ini:



Notifikasi tidak ada *manual entry*

Ini karena Neovim tidak menemukan dokumentasi untuk *tag span* tersebut. Untuk mengatasi ini kita dapat memasang *server LSP* untuk HTML dengan cara menggunakan perintah :**Mason** dan tekan **2** untuk beralih ke *tab LSP*, lalu cari *html-lsp* dan tekan **i** untuk memasangnya. Jalan pintas untuk memasang *server LSP* adalah dengan menggunakan perintah :**LspInstall nama-server**.

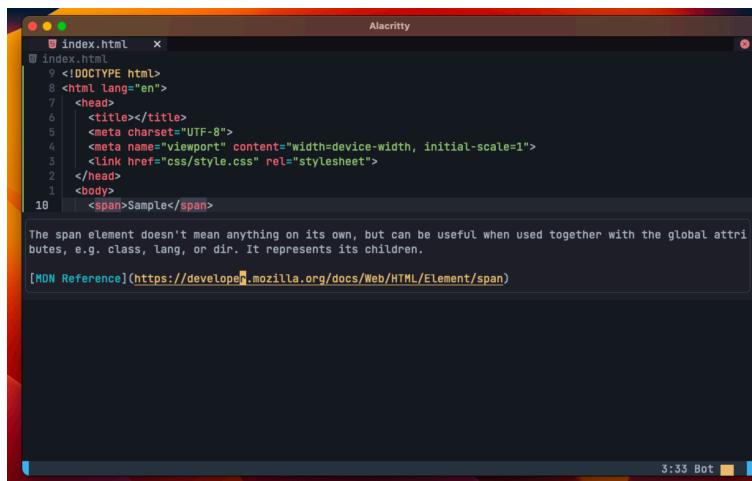


```
mason.nvim
press g? for help
https://github.com/williamboiman/mason.nvim
(1) All (2) LSP (3) DAP (4) Linter (5) Formatter
Language Filter: press <C-f> to apply filter
Installed (0)
No packages.

Installing (1)
  <bo> npm install vscode-langservers-extracted
Available (130)
  ada-language-server
  angular-language-server
  ansible-language-server
  apex-language-server
  arduino-language-server
  asm-lsp
  astro-language-server
  awk-language-server
  bash-language-server
  beancount-language-server
  bicep-lsp
```

### Memasang *server LSP* HTML

Tunggu proses pemasangan hingga selesai. Jika sudah selesai, tutup *floating window* dengan perintah q dan kembali gunakan perintah K pada tag *span* untuk membuka dokumentasi, sekarang seharusnya sudah muncul seperti ini:

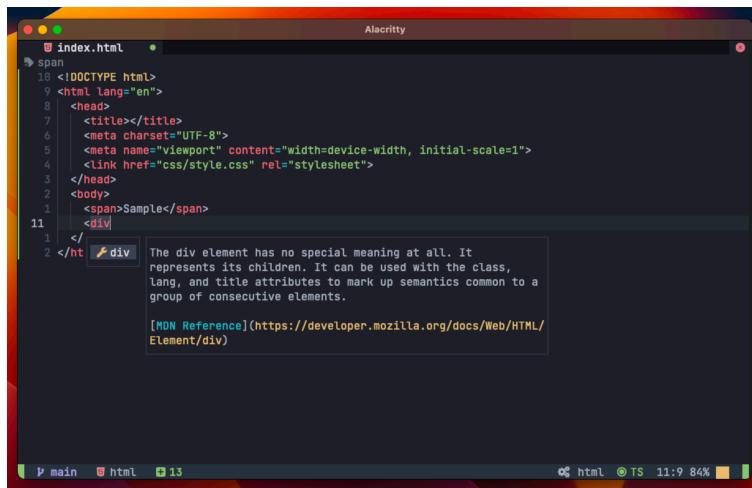


```
The span element doesn't mean anything on its own, but can be useful when used together with the global attributes, e.g. class, lang, or dir. It represents its children.
[MDN Reference](https://developer.mozilla.org/docs/Web/HTML/Element/span)
```

### Fitur dokumentasi LSP

Bila belum muncul, jalankan ulang Neovim terlebih dahulu.

Selain itu, kita juga mendapatkan fitur *autocomplete* untuk HTML, seperti ini:

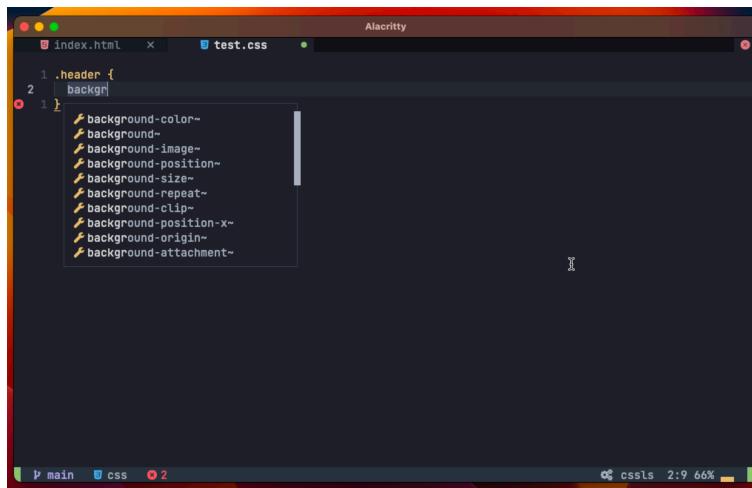


The screenshot shows a terminal window titled "Alacritty" with the file "index.html" open. The code includes a `<div>` element containing the text "Sample". A tooltip is displayed over the `<div>` tag, providing information about the `div` element: "The div element has no special meaning at all. It represents its children. It can be used with the class, lang, and title attributes to mark up semantics common to a group of consecutive elements." Below the tooltip is a link to the MDN Reference: <https://developer.mozilla.org/docs/Web/HTML/Element/div>. The status bar at the bottom shows "html @ TS 11:9 84%".

Fitur autocomplete

Untuk bahasa yang lain, kita dapat menggunakan cara yang serupa dengan sebelumnya. Sebagai contoh, kita akan memasang *server LSP* untuk CSS dengan perintah `:LspInstall cssls` dan tunggu hingga selesai proses pemasangannya.

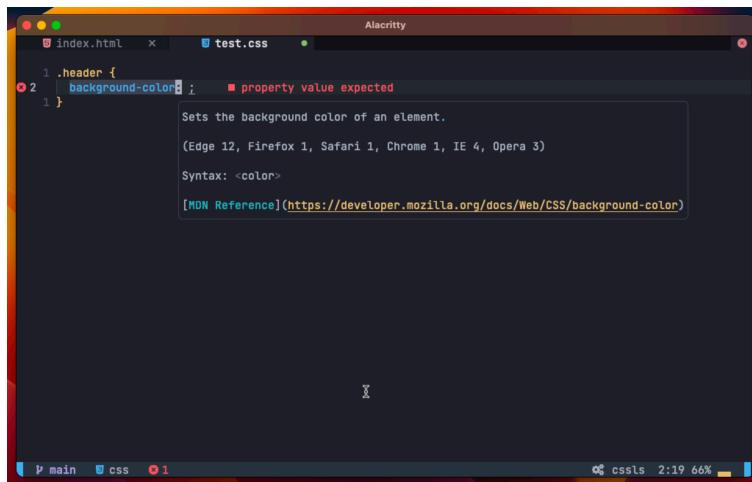
Kita dapat mengujinya dengan membuat *buffer* atau berkas CSS baru, misal seperti ini:



The screenshot shows a terminal window titled "Alacritty" with two buffers: "index.html" and "test.css". The "test.css" buffer contains a CSS rule for ".header" with a partially typed background property. An auto-completion dropdown menu is open, listing various CSS properties starting with "background-", such as "background-color~", "background-", "background-image~", etc. The status bar at the bottom shows "cssls 2:9 66%".

Menguji *autocomplete* untuk CSS

Untuk contoh dokumentasinya:



A screenshot of the Alacritty terminal window. The title bar says "Alacritty". There are two tabs open: "index.html" and "test.css". The "test.css" tab is active and contains the following code:

```
.header { background-color: ; }
```

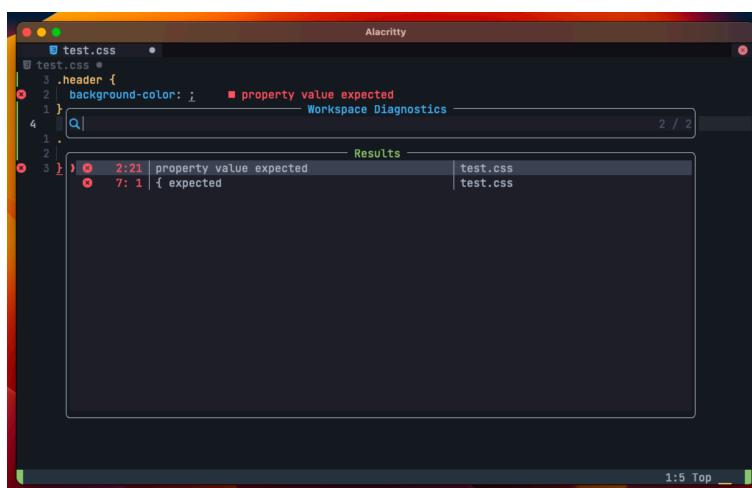
An inline error message "property value expected" is shown at the end of the line. A tooltip box appears over the error, providing information about the "background-color" property:

Sets the background color of an element.  
(Edge 12, Firefox 1, Safari 1, Chrome 1, IE 4, Opera 3)  
Syntax: <color>  
[MDN Reference](<https://developer.mozilla.org/docs/Web/CSS/background-color>)

Dokumentasi *server LSP CSS*

Selain fitur *autocomplete* dan dokumentasi, Neovim juga memunculkan *diagnostic* dari *server LSP* yang sedang digunakan. Seperti pada tangkapan layar di atas yang memberikan *inline error* bertuliskan "*property value expected*". *Diagnostic* dapat berasal dari beberapa sumber, seperti *server LSP* atau *linter*.

Kita dapat melihat semua daftar *diagnostic* yang terdapat di sumber kode kita dengan bantuan Telescope, gunakan perintah :Telescope diagnostics untuk melihatnya.



A screenshot of the Alacritty terminal window. The title bar says "Alacritty". The "test.css" tab is active and contains the same code as before:

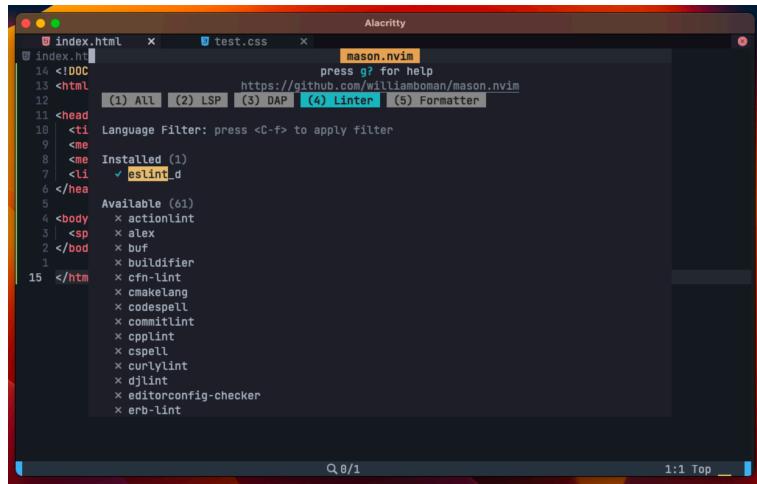
```
.header { background-color: ; }
```

A "Workspace Diagnostics" panel is open on the right side of the screen, titled "Results". It shows the following diagnostic results:

Line	Column	Message	File
3	2:21	property value expected	test.css
3	7:1	{ expected	test.css

Telescope *diagnostics*

Apabila kita ingin mendapatkan *linting* yang lebih lanjut, kita dapat memasang *linter* untuk masing-masing bahasa, seperti *eslint* untuk JavaScript, atau *stylelint* untuk CSS. Kita dapat menggunakan Mason untuk memasang *linter* dengan perintah :Mason. Kemudian tekan 5 untuk beralih ke *tab linter* dan cari *linter* yang diinginkan lalu tekan i untuk memasangnya.



The screenshot shows a terminal window titled "Alacritty" with two tabs open: "index.html" and "test.css". A floating menu titled "mason.nvim" is displayed over the "index.html" tab. The menu includes a URL "https://github.com/williamboman/mason.nvim" and a help message "press g? for help". Below these are five numbered options: (1) All, (2) LSP, (3) DAP, (4) Linter, and (5) Formatter. The option "(4) Linter" is highlighted with a blue border. A scroll bar indicates there are more items below. The main pane of the terminal shows the content of the "index.html" file, which contains basic HTML code. The status bar at the bottom shows "Q 0/1" and "1:1 Top".

Memasang *linter*

Kita juga dapat menggunakan fitur *formatting* untuk memformat kode pada *buffer* yang sedang aktif. Ketika kita menyimpan *buffer*, maka Neovim akan memformatnya dahulu sebelum menyimpan. Jika kita ingin memformatnya secara manual, kita dapat menggunakan perintah :Format pada *buffer* yang sedang aktif.

Tidak semua *server LSP* mendukung *formatting* contohnya *cssls*, kita perlu menggunakan alternatif yang lebih umum, seperti menggunakan alat Prettier. AstroNvim menggunakan *plugin Null-ls* untuk mengelola hal ini, kita dapat menggunakan perintah :Nullls-Install prettierd. Tunggu hingga prosesnya selesai.

The screenshot shows a terminal window titled 'Alacritty'. In the center, there's a configuration interface for 'mason.nvim'. At the top, it says 'mason.nvim' and 'press g? for help'. Below that is a URL: 'https://github.com/williambonan/mason.nvim'. There are five tabs at the top: (1) All, (2) LSP, (3) DAP, (4) Linter, and (5) Formatter. The 'Available' tab is selected, showing a list of available formatters. One item, 'prettierd', is checked with a green checkmark. Other items listed include 'autoflake', 'autopep8', 'beautify', 'black', 'blade-formatter', 'blue', 'buf', 'buildifier', 'cbfmt', 'clang-format', 'cmakeLang', 'csharpier', 'djlint', and 'dprint'. The bottom right corner of the terminal shows the time '12:1 20%'. The overall theme is dark.

Memasang *prettierd*

Sekarang kita sudah dapat melakukan *formatting* pada kode kita dengan cara menyimpannya atau menggunakan cara yang sudah dijelaskan sebelumnya.

Tentu saja kamu perlu menyesuaikan kembali LSP, *linter* dan *formatter* agar sesuai dengan kebutuhanmu dan penjelasan di atas dapat dijadikan referensi. Sungguh begitu luas bila kita membahas semuanya satu per satu.

Fitur-fitur yang dijelaskan sebelumnya datang dari *server LSP* atau alat-alat eksternal lainnya, kemudian konfigurasi AstroNvim mengatur sisanya sehingga kita dapat menggunakaninya dengan mudah. Kendati Neovim mendukung LSP bukan berarti semuanya hanya perlu dipasang dan tiba-tiba jalan sesuai dengan keinginan kita.

Setiap fitur yang terdapat dari masing-masing *server LSP* harus diatur agar kita dapat menggunakaninya, sebagai contoh untuk memunculkan sugesti *autocomplete* dari *server LSP* kita perlu mengintegrasikannya dengan *plugin* seperti CMP. Tentu saja hal ini membutuhkan waktu dan tenaga bila kita lakukan secara mandiri dari awal.

# Penutup

Kita telah mempelajari banyak tentang Vim dan Neovim, termasuk bagaimana menggunakan AstroNvim sebagai *layer IDE* untuk meningkatkan produktivitas dan efisiensi saat mengembangkan perangkat lunak dengan Vim.

Kita telah membahas berbagai fitur dan hal-hal dasar Vim, seperti memahami jenis-jenis mode, perintah, *plugin* hingga *command-line*, serta cara mengkonfigurasi Neovim untuk memenuhi kebutuhan pengembangan kita.

Di masa depan, kita dapat melihat bagaimana teknologi terus berkembang dan mengubah cara kita bekerja. Vim dan Neovim akan terus berkembang untuk memenuhi kebutuhan tersebut, dan kita dapat memanfaatkan fitur-fitur baru yang ditambahkan untuk meningkatkan produktivitas dan efisiensi kita.

Bagaimanapun, buku yang saya tulis ini hanya berupa sebuah pengantar Vim. Saya menyarankan kamu untuk terus belajar dan mencari sumber-sumber belajar yang dapat membantu kamu dalam menggunakan kedua *program* ini secara lebih efektif. Ada banyak sumber belajar gratis dan berbayar yang dapat kita akses, mulai dari dokumentasi resmi, tutorial daring, hingga buku-buku khusus tentang Vim dan Neovim.

Akhir kata, saya ingin mengucapkan terima kasih kepada tim pengembang Vim dan Neovim yang telah bekerja keras untuk membuat dan menjaga kualitas dari kedua *program* tersebut. Selain itu saya juga ingin mengucapkan terima kasih kepada tim yang memelihara AstroNvim dan *plugin-plugin* yang digunakan di dalamnya.

Tidak lupa, saya juga ingin mengucapkan terima kasih kepada kamu yang telah menyelesaikan buku ini dan memperhatikan usaha saya dalam menulisnya. Saya harap buku ini dapat memberikan wawasan dan pengetahuan yang bermanfaat bagimu, dan semoga kita dapat

menggunakan Vim dan Neovim dengan lebih efektif dalam menyelesaikan tugas-tugas pengembangan perangkat lunak kamu.

Selamat melanjutkan petualanganmu di ekosistem Vim!

# Penulis

Muhamad Nauval Azhar adalah seorang *programmer* yang tinggal di Bogor. Ia memulai kariernya sebagai *programmer* sekitar 1 dekade yang lalu, dan sejak itu terus berkembang dan belajar tentang teknologi terbaru di bidang ini.

Selain bekerja, Muhamad juga memiliki hobi membuat proyek sumber terbuka yang bermanfaat bagi masyarakat. Salah satu proyek terfavoritnya yang ia buat adalah Stisla, sebuah *template* admin gratis yang dapat digunakan untuk membangun aplikasi web modern. Stisla telah digunakan oleh ribuan pengembang di seluruh dunia, dan saat ini tersedia di situs web resmi [getstisla.com](http://getstisla.com).

Selain itu, Muhamad juga membuat proyek lain yang bernama array id ([arrayid.org](http://arrayid.org)), sebuah komunitas *online* yang membahas tentang teknologi dan pengembangan perangkat lunak. Di array id, Muhamad dan teman-teman yang lain membagikan berita, tutorial, dan berdiskusi tentang teknologi terbaru untuk membantu para pengembang meningkatkan kemampuan dan pengetahuan mereka.

Muhamad menulis buku Pengantar Vi iMproved dengan tujuan membantu para pembaca yang ingin memulai belajar juga beralih ke Vim dan Neovim, serta menyediakan panduan yang mudah dipahami bagi mereka yang ingin menggunakan kedua *program* ini secara efektif. Ia berharap buku ini dapat memberikan wawasan dan pengetahuan yang bermanfaat bagi para pembaca, dan semoga dapat membantu mereka dalam meningkatkan produktivitas dan efisiensi dalam mengembangkan perangkat lunak.

*Kita perlu membiasakan mencoba alternatif lain untuk melihat potensi pengalaman yang lebih baik. Bukan menutup diri dengan berlagak konservatif.*