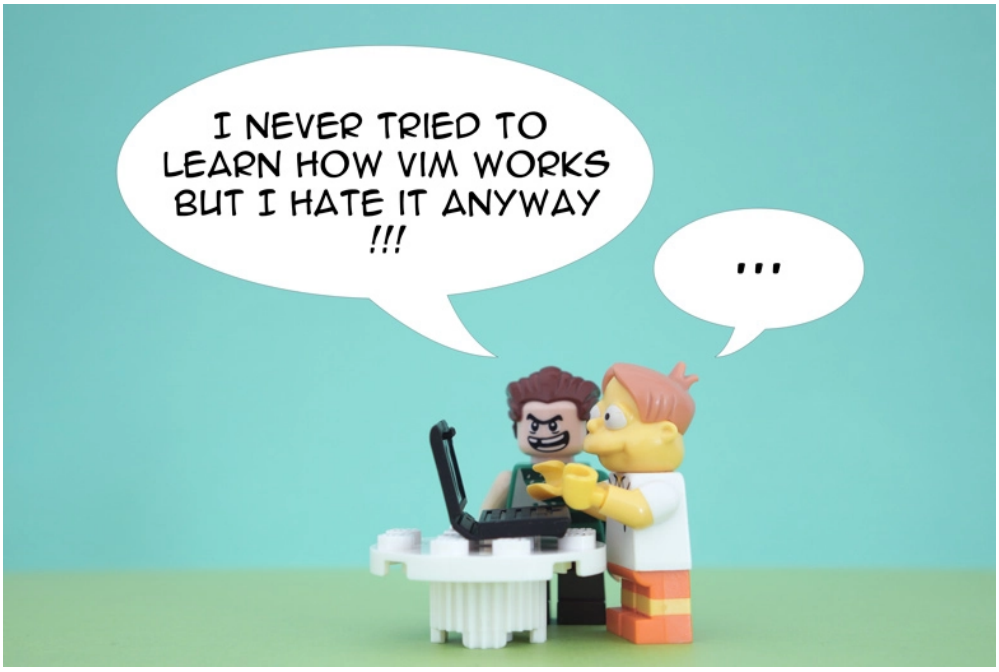
Are Your Ready for Vim? A Beginner Guide



Welcome to the first part of this series of articles to learn Vim!

Vim from the ground up
“Vim is not for me!” shout out Dave, my colleague developer.

It wasn’t the first time I’ve heard this sentence. The result? Me, explaining to Dave and others gathering around my desk, that learning the basics of Vim can be beneficial to anybody:

- Many CLIs use Vim-like key bindings, like Less for example.
- They can easily edit files on remote systems or in docker containers when only Vi (the ancestor of Vim) or Vim is available.
- They can customize Vim like crazy, to match their personal needs and preferences.
- They can run Vim everywhere they want.
- Vim can edit very large text files without slowing down, like huge log files for example.
- They can learn a new and really *fun* way for creating and editing content.

To me, Vim is the gamification of coding.

Still, some developers don't try to understand what Vim is about. At the same time, they have a strong opinion about it. How can somebody judge something without *seriously* trying it?

I can't blame them: I had exactly the same bias years ago. But when I tried to learn to use Vim, when I tried to understand how it works - not only learning two shortcuts randomly - I fall in love.

That's why I would like to share with you today how I learned the very basics of Vim. More specifically, we'll see in this article:

- How to learn quickly good typing techniques. They are essential if you want to unleash Vim's latent power. Even if you don't use Vim, these techniques have many benefits for any developer.
- The different Vim modes. This is one of the main reason why Vim is so crazy.
- The basic Vim shortcuts (keystrokes) for you to be efficient as quickly as possible, with some tips to remember them easily.
- How to speak the Language of Vim.
- What Vim's options are and how to manipulate them.

After that, you'll speak about Vim with experience and confidence, even if it's for saying: "naaaah, Vim is not for me!".

The goal of this article is *not* to replace your IDE with Vim from one day to another. I would recommend to take a step by step approach here.

At the beginning, you can try to use Vim to edit some configuration or other text files. Practicing what you'll learn in this article is the key for you to really understand how Vim works and why it's so popular, even decades after its creation.

Don't think that Vim is hard to learn. It's *easy to learn* to edit any file, but it's *hard to master*. Vim gurus, coding in the Himalaya for hundred of years, can't even pretend knowing everything about Vim. That's

great, because it means that the possibilities of this editor are beyond infinity.

Doubtful? Follow me. Let's dive into the wonderful world of Vim together.

Prerequisites: The Power Is In Your Fingers

When I decided to learn Vim, I wanted to do it right. Vim allows you to forget your hands and let you really focus on the most important thing: what you're writing.

The Mouse: Your False Best Friend

One of the advantage of Vim is to let your hands on the keyboard, without the constant need to grab your mouse.

I see you're afraid: your mouse is like your third hand! It's so useful and easy! Why would you not use it?

Your mouse is a bit like an implant a doctor would have put on your body at a very young age, telling your parent that it's the best device to do something on a computer. You like it because you're deeply used to it, for a very long time.

Ask yourself: why on earth, if the mouse was so perfect, your favorite IDE has 341324 *keyboard shortcuts*? Maybe because using your keyboard is faster? Easier? More efficient? More *comfortable*?

Your mouse is not your best friend. It's just a friend. Your keyboard is the real brother-from-another-mother here. The power comes from it, and Vim is perfect for you to harness and unleash this power.

Note
If you want to build a complete Mouseless Development Environment, you might be interested by this book.
That's why I deeply believe that before learning Vim, you should learn basic typing techniques. The benefits are great:

- You'll type faster and more accurately.
- You'll be able to increase your speed and accuracy over time.
- You won't focus on your keyboard anymore. Not even a bit.

If you already use these techniques, that's great! You can directly go to the next chapter.

Efficient Typing: The Two Rules

We all agree that thinking, for a developer, is more important than knowing how to type. That said, it's still nice to feel in control of your tools. As a developer, the keyboard is one of the most important!

It's very fulfilling to see your typing improving day after days, months after months, even years after years. The room for progression is huge, even if it's pretty quick and easy to learn the basics.

The first rule you need to learn is placing your hand correctly:

The keys `a,s,d,f` and `j,k,l,;` are called the *row keys*. They are the starting points for your hands. From there, you'll be able to grab any other key easily.

You'll notice that there are little bumps on the `f` and `j` keys on your keyboard: they are indicators for you to know where to put your

indexes. When they are at the good position, simply place the other fingers on the other row keys.

The second rule you need to train for: try not looking at your keyboard while you're typing. If you don't remember where a key is, try first to get it without looking, even if it feels random.

I was only typing with two fingers for years. It felt totally foreign to try to follow these rules at first. Now, I could not type differently, mostly because it's more comfortable.

It took me only one to two weeks to learn this new way of typing. It's not because I'm a genius (definitely not), but because it's easy.

The First Week

When you decide to use the two rules I described above, you need to try to follow them *all the time*. We need 100% commitment here. When you surprise yourself using your old (and bad) techniques again, be patient, don't see it as a failure, and come back to the good ones. This is the mandatory part of the learning process.

The first three days are the most difficult. You'll alternate between good and bad technique without even noticing it. You'll do mistakes, and you'll be slow.

Fortunately, at the end of the week, the amount of mistake will decrease, and you will less and less need to watch your keyboards.

The Second Week

You'll notice during the second week the amount of mistakes decreasing even more, and your need to watch your keyboard will disappear.

At the end of the week, you'll see your typing speed already improving. The good feelings of reward will begin to please your sweet brain.

Speed and Accuracy

During your two weeks of initial training, you shouldn't focus on speed or accuracy. Just type, as much as you can, and don't worry about anything else yet, not even the mistakes you make.

After that, you can focus on speed and accuracy: how fast you can type and trying to make as fewer mistakes as possible.

To train these good typing techniques, from the beginning of your learning experience to the end of your life, you can use typing software which can drastically help you.

Here's a list of my favorite ones:

Vim Or Neovim?

With these fundamentals out of the way, let's install Neovim.

Neovim? What's this new weird thing, you might rightfully ask?

Neovim is a refactor of Vim. It's compatible with everything Vim related. I would definitely recommend using it, instead of the regular Vim, since it's optimized out of the box.

Here are the official links for both software:

Because Neovim and Vim are *almost* identical (with a different philosophy), I'll continue to call these two software using the generic term *Vim* in this article.

Vim's Configuration

In Vim, almost everything is configurable. It's insane, I tell you. You can shape your editor according to your specific needs and megalomaniac desires.

Depending on if you use Vim or Neovim, your configuration file should be in the following path by default:

- For Vim: `~/.vimrc`.
- For Neovim: `~/.config/nvim/init.vim` or `~/nvim/init.vim` (depending on the value of the environment variable `$XDG_CONFIG_HOME`).

This configuration file is sweetly named *vimrc*.

Let's open your vimrc with your favorite editor (which might be soon replaced by Vim!) and let's add some basics things. I'll explain later the purpose of them:

```
noremap <Up> <Nop>
noremap <Down> <Nop>
noremap <Left> <Nop>
noremap <Right> <Nop>
```

and

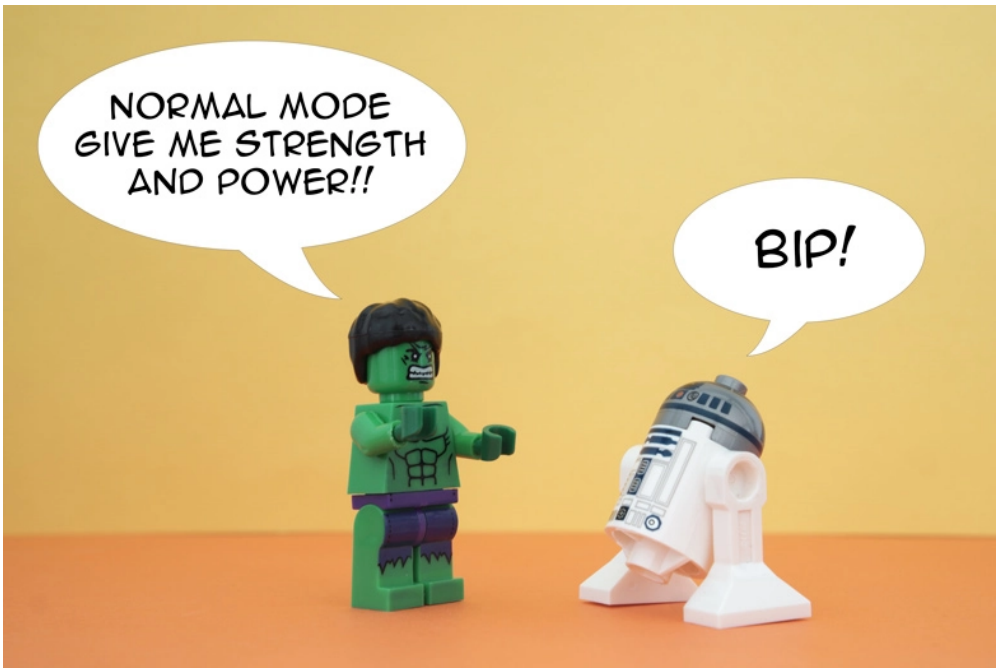
```
set clipboard+=unnamedplus
```

The content of your vimrc and the different plugins for Vim are often written in Vimscript, a programming language specifically created for the editor. Yet, I wouldn't advise you to learn this language, except the

basics to configure Vim. It's quite painful to use. That's why Neovim wants to replace it with Lua, which is a very good idea indeed.

Now, let's launch Vim and, without trying to do anything else, let's see the basics you need for you to get started.

The Modes of Vim



After launching Vim, you will see a welcome screen displaying the very basic commands you can use. This screen will disappear as soon as you begin to type some content.

Vim is not like your standard editor where you can simply type on your keyboard and your content will magically appear on your screen. Try to type `x`, for example: nothing seems to happen.

This is because Vim has *modes*, and you can do different things depending on the mode you're in. To spot easily the different modes I'm speaking about in this article, I'll always write them in uppercase (i.e. NORMAL mode).

Normal Mode

Normally, when you open an editor, you can directly type some content. Not in Vim. Its default mode is NORMAL mode, where you can

edit some existing content.

You can use keystrokes to move where you want and edit what you want: inserting, changing or deleting words, sentences, or even paragraphs. Without using your mouse once!

Think of it as a system of shortcuts which allows you to target exactly what you want to edit. The difference between your default GUI editor and Vim when we speak about shortcuts (or keystrokes) is significant: they make sense in Vim, most of the time. They follow a certain logic, and they are *composable*. That's why Vim is easy to learn.

For example, the shortcut `CTRL+shift+n` to find a file in your favorite GUI editor is difficult to remember because it's difficult to link what you know (opening a file) and what you want to learn (the keystroke).

We will come back to the NORMAL mode later, when we will learn the main Vim's keystrokes.

Insert Mode

Look at your cursor in Vim: it should normally be a square. Now, let's hit our first NORMAL mode keystroke: `i`, for `i`nsert.

What's this dark magic? Your cursor became a pipe! In the bottom left corner, the indicator `--INSERT--` appeared! How impressive! How marvelous! Do you think I'm exaggerating? Yes I do!

Welcome to the INSERT mode.

You can finally type your content in this mode: go ahead, type anything you want. To come back to NORMAL mode and stop inserting, simply hit `ESC` or `CTRL-c`. The cursor becomes a square again, and the `--INSERT--` indicator disappears. Don't be sad, it will come back.

That's how you work with Vim: juggling between NORMAL mode to edit *existing* content and INSERT mode to insert some *new* content.

Visual Mode

There is a third important mode in Vim you'll often use: VISUAL mode. Its goal? Selecting your content. From there, you can modify, edit or copy your selection.

To enter VISUAL mode, you might have guessed it already, you need to type `v` in NORMAL mode. You'll see the indicator `--VISUAL--` appearing in the bottom left corner of your Vim instance. Again, to come back to our default mode, NORMAL mode, press `ESC` or `CTRL-c`.

You can as well select entire lines if you start the visual mode linewise with the keystroke `SHIFT+v`. That's not all: you can start visual mode blockwise, using the keystroke `CTRL+v`. It's pretty useful when you have to deal with tables or lists, for example.

Command-Line Mode

You can compare the COMMAND LINE mode as a menu on a GUI. A big and powerful one.

If you type `:` in NORMAL mode, your cursor will end up automatically at the bottom of Vim. From there, you can type any command you want. These commands are also called *Ex command*.

Here are the most basic ones:

- `:help` to open Vim's help. Arguably the most useful command. This help is insanely complete. If you don't remember how to quit Vim for example, you can type `:help quit`.
- `:q` to quit the current window. If there is only one window (it's the default), you'll finally quit Vim.
- `:q!` to quit without saving. Imagine that you yell at your editor you want to quit, whatever the consequences!
- `:w` to write (save the current file open).
- You can even combine some ex command: `:wq` to write and quit. Or just use `:x`.

- `:e <path>` to edit a file. The path can be absolute or relative.

Now that you know how to access Vim's help, I'll put at the end of each section the help commands you can use to go deeper. For example, here's the ones for this section:

Vim help

- `:help helphelp`
- `:help vim-modes`
- `:help insert.txt`
- `:help visual.txt`
- `:help cmdline.txt`
- `:help write-quite`
- `:help ex-cmd-index`

Don't worry if you don't understand what's written in the help itself, or if it's too daunting. Bear with me in this article, and later, when you're more comfortable with Vim, you can come back to this delightful help.

General Vim Keystrokes

Now that we understand the general concept of Vim and its modes, let's see the most important keystrokes in NORMAL mode you need to be aware of. I encourage you to try them in Vim as you read.

Writing Your Own Cheatsheet

You can find countless cheatsheets on the Internet which will give you as many keystrokes as you want. When I was learning how to use Vim, I used one of them.

At the same time, I was writing *my own cheatsheet*.

Why? Personally, it helped me a lot memorizing these keystrokes. Writing is a powerful way to make the information yours.

Organize your cheatsheet as you like, and use whatever you want (paper, evernote, mindmaps…). For mine, I'm using Joplin, a free and open source note taking system.

Bonus points if you write your cheatsheet… in Vim!

The rest of the article will give you the basics keystroke you need at the beginning. Be aware that Vim is full of subtleties which give you power and efficiency, depending on the context. It's a never ending learning process which makes this editor so interesting, so fun to use, and so rewarding!

Searching

I wrote a whole article about searching in Vim, but, for now, the keystroke `/` should be enough. If you use it, your cursor will end up the bottom of the screen. From there, type your search and press ENTER.

You can go to the next found occurrence by typing `n`. To go to the previous one, use `N`.

You can also use these two keystrokes in NORMAL mode to search the word under the cursor:

- `*` - Search forward.
- `#` - Search backward.

Undo and Redo

What would we do without the essential undo and redo?

- `u` will `u`ndo your last edit.
- `CTRL-r` will `r`edo. You can think of it as you being in control (`CTRL`) of your content.

Insert Mode

We saw previously that the keystroke `i` moved your soul in the reassuring world of INSERT mode.

There are other handy keystrokes to do so, introducing welcome subtleties:

- `i` - `i`nsert content before the current character.
- `a` - insert content `a`fter the current character.
- `A` - insert content `A`fter everything. It will move your cursor to the end of the line and enter INSERT mode.
- `o` - `o`pen a new line, below the current one, and allow you to insert your content.
- `O` - `O`pen a new line above the current one.
- `esc` or `CTRL-c` or `CTRL-[` - bring you back to NORMAL mode if you are in INSERT mode.

Vim help

Vim's Motions

Motions in Vim allow you to move your cursor horizontally (on a line) or vertically.

Note that you can combine motions with a count: if you want to make your motion 6 times for example, you can do `6<your_motion>`. Replace `<your_motion>` with whatever motion you want.

You can use motions in NORMAL mode and VISUAL mode.

Forget the Arrow Keys

To be totally honest with you, this was the hardest part for me: not using the arrow keys to move your cursor.

As I said in the first part of the article, your fingers should be on the row keys. First, for your typing to improve, and second because the Vim's keystrokes you can use in NORMAL mode are all around the row keys. Your hands shouldn't move too much; only your fingers should.

Now, try to reach the arrow keys from the row keys: yes, you need to move your hand! This is definitely not what we want.

That's why, instead of using the arrow keys, we should use the keys `h`,`j`,`k` and `l` to move respectively left, down, up and right.

It's difficult at first: you'll try to use the arrow keys to move your cursor, and more than once. That's why we previously disabled them in the configuration!

How to remember what does `h`, `j`, `k` and `l`?

- `h` moves your cursor to the left, and `l` moves it to the right. It makes sens, since `h` is on the left of the sequence `hjkl`, and `l` is on the right.
- `j` moves your cursor down. You can remember it since `j` looks like an arrow which points down (with a bit of imagination). Another mnemonic method: the key `j` has a little bump at the bottom of the key, which means the cursor will go down.
- `k` is the only letter left, so it has to go up. I always imagine a Ninja Turtle *jumping up*, saying "Kowabunga"! It's not even the good spelling (it would be "Cowabunga") but it works for me. Please, don't judge me.

I see your mind full of questions. Fear not, my friend! I've got you covered for this one, with a revolutionary AAA game everybody will speak about in twenty years. To play it, you *must* use `hjkl`.

If you prefer puzzle games, try this wonderful sokoban. You can use `hjkl` or the arrow keys this time, but try to only use `hjkl`.

Moving Horizontally

Here some basic keystrokes to move quickly on a line in NORMAL mode:

- `w` - move forward to the next word. A word - by default - is a sequence containing letters, digits, or underscores.
- `b` - move back your cursor to the previous word
- `0` - move to the beginning of the current line.
- `^` - move to the first non-blank character on the current line.
- `$` - move to the end of the current line.
- `%` - move to the matching bracket when the cursor is already on a bracket.

You can as well move on a specific character on your line using:

- `f<character>` - find a character after your cursor.
- `F<character>` - find a character before your cursor.
- `t<character>` - move to a character after your cursor.
- `T<character>` - move to a character before your cursor.

After using one of the last four keystrokes, you can move on the character you've chosen with `;` to go forward, and `,` to go backward. A very powerful way to move horizontally!

Moving Vertically

You can move vertically simply by searching the word you want to move on (see above to learn how to search). There are other ways to move vertically:

- `<line_number>G` - move your cursor to the beginning of an arbitrary line. For example, `10G` will move the cursor on line 10.
- `G` - move your cursor to the last line of your file.
- `1G` or `gg` - move your cursor to the first line of your file.

You can as well use different keystrokes to scroll. Here are the very basics:

- CTRL - e - scroll the window downwards.
- CTRL - u - move your cursor upward half a screen.
- CTRL - d - move your cursordownward half a screen.

Vim help

The Language of Vim

In Vim, keystrokes can be seen as “sentences”, describing an action. That’s what I meant when I was saying that the keystrokes are *composable*. I know, it sounds weird, but it’s brilliant.

These “sentences” are so common that you’ll associate easily what you know already (the sentence) by what you need to learn (the keystrokes).

Even better: knowing that Vim has a “keystroke language” will push you to combine them instinctively to do what you need to do, and, in many cases, it will work!

Operators

Operators are the verbs of the Vim language. They need to be combined with motions or text-objects to work. Here are some important operators:

- `d` for delete
- `c` for change
- `y` for yank (copy)

When using the yank operator, you can `put` what you've copied using the keystrokes `p` or `P`.

You can combine operators and motions as follows:

- `d$` will delete from your cursor to the end of line. You can use as well the alias `D`.
- `dgg` will delete everything from the cursor to the beginning of the file.
- `ggdG` will delete everything in the file!

Text Objects

If operators are the verbs, text objects are the nouns. Simply put, a text object is a set of character. In Vim, a `word` is a text object, as well as a `sentence` or a `paragraph`.

For example, you can use operators and text objects as follows:

- `diw` will delete `i`nside the `w`ord. It will delete the current word under the cursor.
- `ciw` will `c`hange `i`nside the `w`ord. It will delete the current word under the cursor and switch to INSERT mode. In short, you… change the word!
- `dip` will delete `i`nside the `p`aragraph.

You can try to *change a word* or *delete a word*, it works as well and introduce some subtleties. I let you find what could be the keystrokes for these!

Vim help

Vim's Options

You can modify Vim's behavior by modifying its options.

You can think of an option as a variable. It can be a boolean (which can be switched on and off), a string, or a number. You can display their values or modifying them using the COMMAND-LINE mode.

If you want to permanently modify some options, assign their new values directly in your vimrc, as we did with the option `clipboard` at the beginning of this article.

Here are the commands you can use:

- `:set no<option>` - Unset the option.
- `:set <option>!` - Toggle the option.
- `:set <option>?` - Return the option's value.
- `:set <option>=<value>` - Set a value `<value>` (string or number).
- `:set <option>+=<value>` - Add the value `<value>` for a number option, append a string `<value>` for a string option.
- `:set <option>-=<value>` - Subtract the value `<value>` for a number option, delete the string `<value>` for a string option.
- `:set <option>&` - Reset the option to its default value.

For example, if you want to display the filetype of the current file open, you can run:

```
:set filetype?
```

Drop the prefix `:` if you want to set these options in your vimrc.

Vim help

The Beginning of the Vim Journey

Congratulations! You've been initiated to the dark knowledge of Vim. Let's summarize what we learned in this article:

- Having proper typing techniques will make you faster, more accurate, and will allow you to focus solely on your content, not on your keyboard or on your hands. You'll feel as well more in control (a good way to lower your stress).
- Write your own cheatsheet as you learn new keystrokes, and try to find your own mnemonics to retain them. It will accelerate your learning process. To remember easily, associate what you already know with what you want to learn.
- You can use a combination of operators, motions, and text objects to edit your text in NORMAL mode.
- You need to use the row keys `h`, `j`, `k` and `l` to navigate in Vim, to really appreciate its comfort, its power, and its fun!
- Try for yourself the keystrokes explained here, for you to memorize them using your muscle memory.
- Vim's options can be switched on, off, toggled, or unset (for booleans). You can as well modify their values (for strings and numbers). It's always interesting to display their values to be sure Vim behave as you expect.

Even if you still don't like using Vim at that point, at least you tried. You'll be able to use it when you're lost on a distant remote server and, more important, you can now claim, legitimately, that you don't like Vim!

What's next in your Vim learning journey?

- You can read the second part of this series of article, Vim for intermediate users.
- You should go through `vimtutor`. If you use Neovim, you just have to execute the ex command `:Tuto`. If you use Vim, simply type `vimtutor` in your shell.
- I would heavily recommend reading the book Practical Vim, from Drew Neil (see below). You'll learn *a lot* from it.
- Vim is a game: the goal is to use as few keystrokes as you can to accomplish what you want to do. In that spirit, Vim Golf can be pretty fun.

If you didn't become a Vim Hater, you're now on the path to become a Vim Master.

Learning to play vim

If you like this series, I'm currently writing an ambitious book about Vim with many more tips!

Related Sources