- Expert Mentors
- Expert Mentors
  - Web Programming
    Web Programming
    Ruby
    JavaScript
    AngularJS
    Python
    PHP
    HTML/CSS
    jQuery
    Ruby on Rails
    Django
    Node.js
    CoffeeScript
    Ember.js
    Backbone.js
    Meteor
    Flask
    Scala
  - Code
    Code
    Ruby
    JavaScript
    PHP
    Python
    Java
    C#
    C++
    Go
    C
    .NET
    Haskell
    Perl
    Erlang
    Matlab
    Debugging
    Crystal
  - Mobile App Programming
    Mobile App Programming
    iOS
    Android
    Swift
    PhoneGap
    Cordova
    Ionic
    Titanium
    Sencha
  - Design / UX
    Design / UX
    HTML/CSS
    CSS
    Sass
    Twitter bootstrap
    Famo.us
    Kendo UI
    Responsive design
    Foundation
    Photoshop
  - Database / Operations
    Database / Operations
    Server
    DevOps
    MySQL
    SQL
    MongoDB
    Hadoop
    Apache
    Linux
    AWS
    Heroku
    Database
    Security
    Azure
  - Development Process / Tools
    Development Process / Tools
    Git
    DevOps
    WordPress
    Drupal
    Joomla
    SEO
    Vim
    Bower

    Machine learning
    Xcode
    Jenkins
  -
  - Top Developers
    Top Developers
    Ruby
    Javascript
    AngularJS
    Python
    iOS
    Swift
    PHP
    Java
- Free Tutorials
- Free Tutorials
  - Code
    Code
    Ruby on Rails
    JavaScript
    AngularJS
    Python
    iOS
    Android
    PHP
    Swift
    Ember.js
    Node.js
    Java
    Famo.us
    Kendo UI
    Go
    Haskell
    Rust
    Corona
    React
    C#
    Meteor
    C++
    Data Science
    R
  - Web Design / UX
    Web Design / UX
    CSS
    HTML/CSS
    Kendo UI
    Design
  - Development Process and Tools
    Development Process and Tools
    WordPress
    Git
    Keen IO
    Bower
    Magento
    RESTful
    Tmux
    Development process
  - Deployment / Operations
    Deployment / Operations
    SQL
    Bower
    DevOps
    Linux
    Docker
    MongoDB
    Spark
  - Startups
  -
  - Free Office Hours
  - AMA for Developers
  - Quick Tips
- Live Classes
  - React
  - AngularJS
  - Ruby on Rails
  - Python
  - Swift
  - Android
  - Idea to MVP
  - ES6
- Live Classes
- Monthly Plans
- Hire
- How it Works
- **Become a Codementor**

}

- Sign Up
- Sign In

0

Learning Center / Learn Deployment / Operations / Learn Linux / Linux Tutorials

Configuring Your Linux for Development With Zsh, Tmux, and Vim

Luke Murphy ● Linux ● October 14th 2014

Markdown

<p><img class="alignright" src="https://www.gravatar.com/avatar/47e959978ed32592c137c9211f

Codementor Luke Murphy is a software engineer by trade. He spent most of his professional career working with the Python programming language, and likes to write programs in Haskell and Clojure.

As a developer who has been using Linux for as long as he can remember, Luke would like to share his own hacks on how to best set up a Linux machine for development.

Purpose

This tutorial is going to be about setting up a development environment on a Linux machine. Have you ever watched a coding tutorial and saw a really slick editing interface with terminal split screens and a gorgeous color scheme? Well, I am not only going to show you how to configure and setup something impressive but I will also give you the insight into how to configure it as you see fit.

After I cover each tool and the general idea of how to configure it, I will leave a link to a fully customised configuration file in which you will be able to experiment and test in whatever manner you see fit. The benefit is that you will know exactly what to do and how to do it.

Conventions

A quick note on the conventions used in this tutorial. I will be writing a lot of shell commands, so when you see something like:

```
$ echo "Hello, World!"
```

It means you should crack open a terminal and run the command, but you shouldn't include the `$`, that symbol represents the shell prompt! You will also see things like this:

```
# Hi, I am something new
$ echo "Hello, World!"
```

The text you see following the # symbol is a terminal comment. It is included solely for extra contextual help, you don't need to do anything with those. Fine, with that out of the way, let's talk shop.

Tools of the trade

Let's talk about the tools you are going to be using.

Zsh

When you open up a terminal on your machine, you are most likely going to find that you are running a Bash shell. In case you are not sure, you can run:

```
$ echo $SHELL
```

Bash is a nice default but we're not here for defaults, we're here for the best we can get and that means `zsh` and all the bells and whistles that come along with it. In a nutshell, Zsh gives us a much more user friendly experience on the command line, it gives us things like better auto-completion, Vim key bindings and smart guesses when you write a command wrong. `zsh` has a huge user community behind it and those smart people have added many, many customisations. We can discover those as we go.

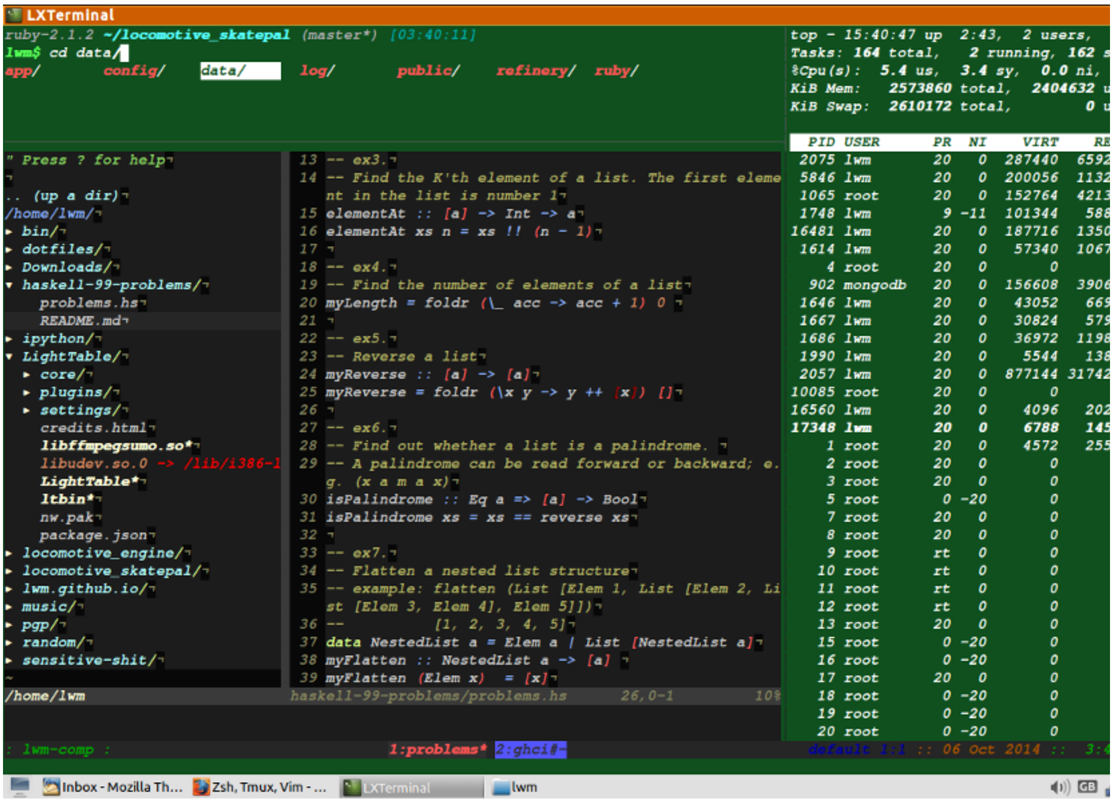
Tmux

Tmux is a terminal multiplexor. Sounds scary right? Well, it isn't at all. Tmux is a tool that allows you to split your terminal screen into any layout you want as well as run multiple terminal instances that are easily accessible. Tmux has become the go-to tool for some of the biggest neckbeards out there.

Vim

`vim` is a text editor and if you haven't heard of it before, well, you're either going to love it or hate it. I do hope you will love it of course as it is an extremely versatile, customisable and powerful text editor. Now, for the sake of diversity, there is another extremely powerful text editor out there called `Emacs` which I would suggest you look into in your spare time and decide whether you would like to learn it. Yes, I said learn, `vim` and `Emacs` have an extremely steep learning curve in terms of how they allow you to manipulate text. I chose `vim` and I will be providing links, tips and customisations to get you where you need to be ASAP.

Here is a preview of the configuration you will end up with after all your hard work:



```
ruby-2.1.2 ~/locomotive_skatepal (master*) [03:40:11]
lwm$ cd data/
app/        config/     data/       log/        public/     refinery/  ruby/
```

```
top - 15:40:47 up  2:43,  2 users,
Tasks: 164 total,   2 running, 162 s
%Cpu(s):  5.4 us,  3.4 sy,  0.0 ni,
KiB Mem:   2573860 total,  2404632 u
KiB Swap:  2610172 total,        0 u

  PID USER      PR  NI    VIRT    RE
 2075 lwm       20   0  287440  6592
 5846 lwm       20   0  200056  1132
 1065 root      20   0  152764  4213
 1748 lwm        9 -11  101344   588
16481 lwm       20   0  187716  1350
 1614 lwm       20   0   57340  1067
    4 root      20   0       0
  902 mongodb   20   0  156608  3906
 1646 lwm       20   0   43052   669
 1667 lwm       20   0   30824   579
 1686 lwm       20   0   36972  1198
 1990 lwm       20   0    5544   138
 2057 lwm       20   0  877144 31742
10085 root      20   0       0
16560 lwm       20   0    4096   202
17348 lwm       20   0    6788   145
    1 root      20   0    4572   255
    2 root      20   0       0
    3 root      20   0       0
    5 root       0 -20       0
    7 root      20   0       0
    8 root      20   0       0
    9 root      rt   0       0
   10 root      rt   0       0
   11 root      rt   0       0
   12 root      rt   0       0
   13 root      20   0       0
   15 root       0 -20       0
   16 root       0 -20       0
   17 root      20   0       0
   18 root       0 -20       0
   19 root       0 -20       0
   20 root       0 -20       0
```

```
" Press ? for help¬
¬
.. (up a dir)¬
/home/lwm/¬
▸ bin/¬
▸ dotfiles/¬
▸ Downloads/¬
▾ haskell-99-problems/¬
    problems.hs¬
    README.md¬
▸ ipython/¬
▾ LightTable/¬
  ▸ core/¬
  ▸ plugins/¬
  ▸ settings/¬
    credits.html¬
    libffmpegsumo.so*¬
    libudev.so.0 -> /lib/i386-l
    LightTable*¬
    ltbin*¬
    nw.pak¬
    package.json¬
▸ locomotive_engine/¬
▸ locomotive_skatepal/¬
▸ lwm.github.io/¬
▸ music/¬
▸ pgp/¬
▸ random/¬
▸ sensitive-shit/¬
~
/home/lwm
```

```
-- ex3.¬
-- Find the K'th element of a list. The first eleme
nt in the list is number 1¬
elementAt :: [a] -> Int -> a¬
elementAt xs n = xs !! (n - 1)¬
¬
-- ex4.¬
-- Find the number of elements of a list¬
myLength = foldr (\_ acc -> acc + 1) 0 ¬
¬
-- ex5.¬
-- Reverse a list¬
myReverse :: [a] -> [a]¬
myReverse = foldr (\x y -> y ++ [x]) []¬
¬
-- ex6.¬
-- Find out whether a list is a palindrome. ¬
-- A palindrome can be read forward or backward; e.
g. (x a m a x)¬
isPalindrome :: Eq a => [a] -> Bool¬
isPalindrome xs = xs == reverse xs¬
¬
-- ex7.¬
-- Flatten a nested list structure¬
-- example: flatten (List [Elem 1, List [Elem 2, Li
st [Elem 3, Elem 4], Elem 5]])¬
--             [1, 2, 3, 4, 5]¬
data NestedList a = Elem a | List [NestedList a]¬
myFlatten :: NestedList a -> [a] ¬
myFlatten (Elem x)  = [x]¬
haskell-99-problems/problems.hs      26,0-1          10%
```

```
: lwm-comp :                1:problems* 2:ghci#-          default 1:1 :: 06 Oct 2014 :: 3:4
```

Prerequisites and Assumptions

A final comment before diving in head first is that I am making some small assumptions about your setup. They are the following:

- You're computers operating system is Debian based.
  - 'Debian based' in simple terms simply means that you are using Ubuntu or a variant like Kubuntu or Lubuntu. The reason for this is that we need to be using the same package manager. Also, things are a lot more user friendly on the Ubuntu side of life.
  - You have a few hours to spare.

That's it! OK, but you're going to need some things installed to get up and running. Here are the prerequisites:

- Git

Did I say some things? I lied, you just need `Git.` What is `Git?` Well, it's a pretty big deal but slightly out of scope right now. All you need to know right now is that it is a tool for grabbing source code. You can install Git by opening up a terminal and running the following command. Actually, while we're at it, let's make sure we have an updated list of packages.

```
# Update our package list
$ sudo apt-get update

# Install Git
$ sudo apt-get install git
```

Zsh

Let's get started. We're going to install `zsh` and kit you out with a decent terminal. Go ahead and run:

```
$ sudo apt-get install zsh
```

That was easy. You now need to switch your default shell from whatever you have now to `zsh`, run:

```
$ chsh -s /bin/zsh
```

Close that terminal and open another and you should be looking at a lovely new `zsh` prompt. We're not done yet, remember all those customisations I talked about? In order to give us a head start let's get the community managed `zsh` framework called `oh-my-zsh`. Head over to the Github page for a look at the nice shiny graphics. OK, let's get it installed.

```
$ git clone git://github.com/robbyrussell/oh-my-zsh.git ~/.oh-my-zsh
```

Now that we have `oh-my-zsh` installed, we can customise as much as we want, let's start by setting up our `.zshrc`. The `.zshrc` is a configuration file that is read by `zsh` every time you open up a new terminal session. In other words, we can put all our customisations in this file and it will be run in the background for us. Let's create that file:

```
# take us back to our home directory
$ cd

# create an empty file called '.zshrc'
$ touch .zshrc
```

Now, I am going to assume you don't have `vim` installed yet, so we can use a simple text editor that comes as default: `nano`.

```
$ nano .zshrc
```

You can type in text simply like with any text editor you are used to. Let's start off by choosing a theme. Head over to oh-my-zsh themes Choose one you like (take a look at "3den", that is the one I use), and then add the following line to your .zshrc

```
ZSH_THEME="3den"
```

Save the file by hitting `Ctrl-x`, and then `y` to confirm that you want to write to the file. Close that terminal and open another up. Brilliant, you should have a new and shiny prompt. I'll take this moment to point out why I use the `3den` theme. `3den` gives me `Git` and `Ruby` information on the command line. It also gives me my current working directory and the time. The colors aren't that bad either. Feel free to experiment with many different themes!

When you want to customise you `zsh` installation, you'll mostly be working in your `.zshrc` file. I am going to list some more things you might want to include in your `.zshrc`, feel free to test them, use them or not. Remember, this is your setup!

```
# want your terminal to support 256 color schemes? I do ...
export TERM="xterm-256color"

# if you do a 'rm *', Zsh will give you a sanity check!
setopt RM_STAR_WAIT

# allows you to type Bash style comments on your command line
# good 'ol Bash
setopt interactivecomments

# Zsh has a spelling corrector
setopt CORRECT
```

That is enough for now, we'll come back to add more later on. Let's move onto `Tmux`.

Tmux

Grab Tmux with the following:

```
$ sudo apt-get install tmux
```

Now, `Tmux` can be a tricky one to figure out at the beginning, but bear with me, we will get there. I have customised my installation with the help of the very great book Tmux – Productive mouse-free development which I totally recommend. Let's not bore you with the defaults, (although you can find them here) go ahead and create the `Tmux` configuration file, called `.tmux.conf`:

```
$ cd
$ nano .tmux.conf
```

Add the following lines:

```
# set Zsh as your default Tmux shell
set-option -g default-shell /bin/zsh

# UTF is great, let us use that
set -g utf8
set-window-option -g utf8 on

# Tmux should be pretty, we need 256 color for that
set -g default-terminal "screen-256color"


# Tmux uses a 'control key', let's set it to 'Ctrl-a'
# Reason: 'Ctrl-a' is easier to reach than 'Ctrl-b'
set -g prefix C-a
unbind C-b

# command delay? We don't want that, make it short
set -sg escape-time 1

# Set the numbering of windows to go from 1 instead
# of 0 - silly programmers :|
set-option -g base-index 1
setw -g pane-base-index 1


# Allow us to reload our Tmux configuration while
# using Tmux
bind r source-file ~/.tmux.conf \; display "Reloaded!"

# Getting interesting now, we use the vertical and horizontal
# symbols to split the screen
bind | split-window -h
bind - split-window -v
```

OK, `Ctrl-x` to save the file and then run:

```
# the '-s' flag specifies a name (we use to attach to it later on)
$ tmux new -s myfirsttmux
```

Great, you see your lovely `Zsh` terminal, now let's play around with those bindings you set in your `.tmux.conf`. Try and run them all in sequence and you will have gotten the basics of `Tmux` in a few minutes!

```
# split the screen in half
Ctrl-a |

# jump over to the right hand split
Ctrl-a

# split that right hand side pane in half
Ctrl-a -

# jump down to that lower pane
Ctrl-a

# Close the window
Ctrl-d

# Open up another window
Ctrl-a c

# Go to the next window
Ctrl-a n

# disconnect from Tmux
Ctrl-a d

# Check what Tmux sessions are running
$ tmux ls

# Attach back into the session
$ tmux attach -t myfirsttmux

# Escape and kill session
Ctrl-a d
$ tmux kill-session -t myfirsttmux
```

OK, lots of information there but you get the gist of it, You can split vertically and horizontally to your hearts content! You also have the ability to reload your `Tmux` configuration file whilst still in a `Tmux` session, so you can add a line, then hit

```
Ctrl-a r
```

Et voilà, you have your new configuration options loaded. Now, I mentioned I would be leaving a link to a fully customised configuration file, and I am going to keep my promise, check out my personal tmux config file . Remember, don't just drop it all in at once, incrementally add options and test them by reloading your config!

Close Tmux, get back to a vanilla terminal and let's move on. You've got the basics here.

Vim

Here comes a big one. Time to get `vim` installed. Run the following:

```
# remove any Vim cruft that might already be on your system
sudo apt-get remove vim vim-runtime gvim  vim-tiny vim-common vim-gui-common

# install vim-nox, a nice Vim starter package
sudo apt-get install vim-nox
```

Go ahead, type:

```
# enjoy ...
$ vimtutor
```

OK, everything should be working fine. Now, like I stated before, `vim` requires some determination to learn, so I suggest that you go read the following resources now or in your own leisure:

- vimcasts.org/
- http://vim-adventures.com/
- www.vimgenius.com/

WARNING: If you don't read about `vim` and how to use it, or take the time to learn some basic keys the following section is going to be quite confusing. I warned you!

Now that you are up to date on what `vim` is all about, let's customise it! Just like our other tools, `vim` has a configuration file, called the `.vimrc`. Create it and let's get editing:

```
# oooh aren't we cool, editing Vims config with Vim
# very meta of us.
$ vim .vimrc
```

So, some words of wisdom, configuring `vim` can be a life's work and I certainly don't hope to fit it all in in this already lengthy post, however, I will point you in the right direction! You can drop a few customisations in just like in the `.tmux.conf`. So, in your `.vimrc`:

```
# give us 256 color schemes!
set term=screen-256color
```

```
# give us nice EOL (end of line) characters
set list
set listchars=tab:▸\ ,eol:¬
```

Save the file and let’s leave that for now. I am now going to show you how to install plugins. Vim plugins are a fantastic way to customise your `vim` installation with some new functionality. It used to be hard to do, but now `vim` has a package manager (actually, more than one!) called Vundle. Let’s go ahead and add a nice new color scheme to `vim`. We need to install `Vundle`, our package manager, so go ahead with:

```
$ cd

# we try to create the '.vim' folder if it doesn't already exist
# along with the 'bundle' folder. Vim + Vundle expect these to exist
$ mkdir -p .vim
$ mkdir -p .vim/bundle

# install Vundle into our .vim folder
git clone https://github.com/gmarik/Vundle.vim.git ~/.vim/bundle/Vundle.vim
```

Now, let’s add `Vundle`‘s config lines to our `.vimrc`. Put these lines right at the top of your `.vimrc`:

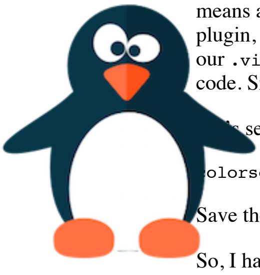
```
# mandatory defaults
set nocompatible
filetype off
set rtp+=~/.vim/bundle/Vundle.vim
call vundle#begin()

# our plugins
Plugin 'gmarik/Vundle.vim'       # vundle
Plugin 'flazz/vim-colorschemes'  # nice colors!
```

Now, save the file and on the command line run:

```
vim +PluginInstall
```

You should see some installing happening. This is good. When everything is finished you will now have access to lots and lots of `vim` color schemes! Head over to the Github page and have a look through the which color scheme you might like.

`Vundle` is simple to figure out, it simply takes Github repositories, clones them (via Git!) for you into the `.vim/bundle` folder and then all you have to do is read the documentation to figure out how the make the plugin in question work (this normally means adding something to your `.vimrc`). Normally, you can find these plugins on Github. If you notice, for the color scheme plugin, the URL was: https://github.com/flazz/vim-colorschemes and we added `Plugin 'flazz/vim-colorschemes'` to our `.vimrc`. Well, this is how it is done, we take the user name and repository url and that is how `Vundle` knows how to get the code. Simple.



’s set a nice color scheme, open your `.vimrc` file again and add the following at the bottom:

```
colorscheme wombat256
```

Save the file and open `vim` again to see your nice new color scheme.

So, I have covered the basics of `vim` customisation but got you nowhere nearly as near to the point where you have all the bells and whistles. Well, don’t worry, I haven’t abandoned you, I provide to the reader a fully equipped `.vimrc` which I currently and is heavily annotated so that you can learn as you go! Check my .vimrc here

WARNING: It is very important that you incrementally add plugins and configuration options to your `.vimrc` and not just paste the whole thing in. You will get very confused, very fast and will not learn anything. Take a plugin, or a configuration line, add it to your `.vimrc` and test it out. You will gain a better understanding and your `vim`-fu will increase ten-fold.

Zsh revisited

Now that you have your `Tmux` and `vim` configured, let’s check some more cool things that we can add to `Zsh`. I’ll just go ahead a list them:

```
# make sure that if a program wants you to edit
# text, that Vim is going to be there for you
export EDITOR="vim"
export USE_EDITOR=$EDITOR
export VISUAL=$EDITOR

# ooh, what is this? Aliases?
source .oh-my-zsh/lib/alias.zsh
```

Yep, that is correct, we can use aliases to save keystrokes and not remember all those long lists of commands. Every time `zsh` runs, these aliases will be sourced and available for your use. Open up your `.oh-my-zsh/lib/alias.zsh`, and drop in the following:

```
# aliases for Tmux
alias tmux='tmux -2'
alias ta='tmux attach -t'
alias tnew='tmux new -s'
alias tls='tmux ls'
alias tkill='tmux kill-session -t'

# convenience aliases for editing configs
alias ev='vim ~/.vimrc'
alias et='vim ~/.tmux.conf'
alias ez='vim ~/.zshrc'
```

I hope you see the value in these. As promised, here is my current `zsh` configuration files, my .zshrc and my aliases.

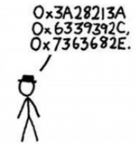
Conclusion

Contents

Phew. We covered a lot there. By now I hope I have ‘taught you how to fish’ and not just given you the fish. You have a powerful editor, a terminal multiplexor and an advanced shell. In all my professional career these 3 tools have enabled me to do anything I wanted from building web site front-ends to hacking on +5,000 line Python projects. Take your time, learn these tools and you will be well equipped to be the neckbeard you always wanted to be.

Bonus

- Automate Tmux – Tmuxinator
- Great Vim book – Practical Vim
- Why Zsh is cool – Zsh, cooler than your shell



Need Luke’s help? Book a 1-on-1 session!

View Luke’s Profile

or join us as an expert mentor!

Want to contribute to Codementor's tutorials? Write for Us!

Get Notified of New Linux Tutorials

sign up to receive our weekly newsletters

Email address Sign Up Now

Related Tutorials

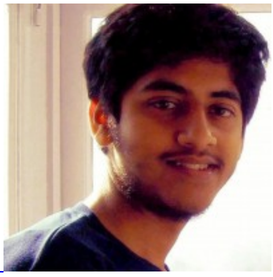
Linux Quick Tip: Adding GitHub Keys as Authorized Keys Vyacheslav

Wrapping a LAMP project into Vagrant with Ansible Vyacheslav

Preparing & Securing Your Ubuntu Box for Deployment Vyacheslav

Getting Started with Cassandra and Spark Sheena

Getting a Development Environment Set Up Before You Finish Your Coffee Vyacheslav

Setting up an Android build server (Ubuntu 13.10 and above)  Arnav Gupta

4 Comments Codementor 1 Login

Recommend Share Sort by Best

Join the discussion…

No · 6 months ago

Oh-My-Zsh has a Tmux plugin that sets up some very similar aliases for you already.

There is also a Vundle plugin that automatically pulls down vundle and sets it up for you. It also sets up a few aliases for running PluginInstall and Clean in vim.

There is a Vi-Mode plugin that that is supposed to make controlling zsh similar to Vim. I haven't used it yet.

Reply · Share ›

Receive New Tutorials GET IT FREE

Abdullah Al Mamun · 7 months ago

Link missing on Why Zsh is cool – Zsh, cooler than your shell

Reply · Share ›

אמריק צוריאל · a year ago

I loved the tutorial, by the way in the configuration of .vimrc, you should use (") instead of (#) ... I think.

Reply · Share ›

F Scott Fitgerald · 3 months ago

Start with 1 instead of zero. Are you retarted!!!!

Reply · Share ›

Subscribe Add Disqus to your site Privacy

Get Notified of New Tutorials

Sign up to receive our weekly newsletters

Email address SIGN UP NOW

codementor

Receive New Linux Tutorials

Email address Sign Up Now
No I don't want free resources from expert developers.

Expert Help
TOPICS
Web Programming Code Mobile App Programming Design / UX Database / Operations Development Process / Tools View All
POPULAR CATEGORIES
Javascript AngularJS Ruby on Rails Python Android iOS C#
Learning Center
TOPICS
Code Web Design / UX Development Process and Tools Deployment / Operations View All
RESOURCES
AMA for Developers Office Hours Javascript Frameworks Codementor Monthly Tutorials Tips Tutors
Company
INFO
Become a Codementor How It Works Codementor for Business Team Training Success Stories Open Source Software

Contact Us FAQ Jobs Blog Write for Us
Codementor
Instant 1:1 help from expert developers

Terms of Service

Sign Up and Get Help Now

Codementor your instant 1:1 expert mentor helping you in real time.

Have an account? Sign In
Want to become a Codementor?
Email address
Sign Up
By clicking the button, you agree to the terms