filename: tmux_conf-settings-and-customisations-multif_20180302.txt
http://www.hamvocke.com/blog/a-guide-to-customizing-your-tmux-conf/

Making tmux Pretty and Usable - A Guide to Customizing your tmux.conf

In my previous blog post I gave a quick and easy introduction to tmux and explained how to use tmux with a basic configuration.

If you've followed that guide you might have had a feeling that many people have when working with tmux for the first time: "These key combinations are really awkward!". Rest assured, you're not alone. Judging from the copious blog posts and dotfiles repos on GitHub there are many people out there who feel the urge to make tmux behave a little different; to make it more comfortable to use.

And actually it's quite easy to customize the look and feel of tmux. Let me tell you something about the basics of customizing tmux and share some of the configurations I find most useful.

Customizing tmux

Customizing tmux is as easy as editing a text file. tmux uses a file called tmux.conf to store its configuration. If you store that file as ~/.tmux.conf (Note: there's a period as the first character in the file name. It's a hidden file) tmux will pick this configuration file for your current user. If you want to share a configuration for multiple users (e.g. if you should feel the urge to start tmux as super user (please think about this carefully!)) you can also put your tmux.conf into a system-wide directory. The location of this directory will be different accross different operating systems. The man page (man tmux) will tell you the exact location, just have a look at documentation for the -f parameter.

Less awkward prefix keys

Probably the most common change among tmux users is to change the prefix from the rather awkward C-b to something that's a little more accessible. Personally I'm using C-a instead but note that this might interfere with bash's "go to beginning of line" command^1. On top of the C-a binding I've also remapped my Caps Lock key to act as Ctrl since I'm not using Caps Lock anyways. This allows me to nicely trigger my prefix key combo.

To change your prefix from C-b to C-a, simply add following lines to your tmux.conf:

```
# remap prefix from 'C-b' to 'C-a'
unbind C-b
set-option -g prefix C-a
bind-key C-a send-prefix
```

Sane Split Commands

Another thing I personally find quite difficult to remember is the pane splitting commands. I mean, seriously? " to split vertically and % to split horizontally? Who's supposed to memorize that? I find it helpful to have the characters as a visual representation of the split, so I chose | and - for splitting panes:

```
# split panes using | and -
bind | split-window -h
bind - split-window -v
unbind '"'
unbind %
```

Easy Config Reloads

Since I'm experimenting quite often with my tmux.conf I want to reload the config easily. This is why I have a command to reload my config on r:

```
# reload config file (change file location to your the tmux.conf you want to use)
bind r source-file ~/.tmux.conf
```

Fast Pane-Switching

Switching between panes is one of the most frequent tasks when using tmux. Therefore it should be as easy as possible. I'm not quite fond of triggering the prefix key all the time. I want to be able to simply say M-<direction> to go where I want to go (remember: M is for Meta, which is usually your Alt key). With this modification I can simply press Alt-left to go to the left pane (and other directions respectively):

```
# switch panes using Alt-arrow without prefix
bind -n M-Left select-pane -L
bind -n M-Right select-pane -R
bind -n M-Up select-pane -U
bind -n M-Down select-pane -D
```

Mouse mode

Although tmux clearly focuses on keyboard-only usage (and this is certainly the most efficient way of interacting with your terminal) it can be helpful to enable mouse interaction with tmux. This is

especially helpful if you find yourself in a situation where others have to work with your tmux config and naturally don't have a clue about your key bindings or tmux in general. Pair Programming might be one of those occasions where this happens quite frequently.

Enabling mouse mode allows you to select windows and different panes by simply clicking on them. I've found that this might mess a little bit with simply selecting text in your terminal using the mouse. You might experience the same, depending on your environment. So you should consider if this configuration is something that's worth it for your use case:

```
# Enable mouse control (clickable windows, panes, resizable panes)
set -g mouse-select-window on
set -g mouse-select-pane on
set -g mouse-resize-pane on
```

Update for tmux 2.1:
As Jon Lillie pointed out in the comments, mouse mode has been [9]rewritten in tmux 2.1. Once you are on tmux 2.1 (or later) you can activate the new mouse mode with a single command:

```
# Enable mouse mode (tmux 2.1 and above)
set -g mouse on
```

The new mode is a combination of all the old mouse options and fixes the text selection issues as well.

Stop renaming windows automatically
I like to give my tmux windows custom names using the , key. This helps me naming my windows according to the context they're focusing on. By default tmux will update the window title automatically depending on the last executed command within that window. In order to prevent tmux from overriding my wisely chosen window names I want to suppress this behavior:

```
# don't rename windows automatically
set-option -g allow-rename off
```

Changing the look of tmux
Changing the colors and design of tmux is a little more complex than what I've presented so far. You'll want tmux to give a consistent look which is why you most likely have to change the looks of quite a lot of elements tmux displays. This is why changes to the design often result in plenty of lines in your config. I can only recommend to put these into their own identifiable section within your tmux.conf to be able to change this block of config without accidentaly ripping out some of your precious custom key bindings. I'm using comments, starting with a # character to make it visible where the design changes start.

Credit where credit is due: I did not create the following design. [10]/u/dothebarbwa was so kind to publish it on /r/unixporn so it's his effort and all thanks have to go out to him. Thanks!

<code>

```
######################
### DESIGN CHANGES ###
######################

# panes
set -g pane-border-fg black
set -g pane-active-border-fg brightred

## Status bar design
# status line
set -g status-justify left
set -g status-bg default
set -g status-fg colour12
set -g status-interval 2

# messaging
set -g message-fg black
set -g message-bg yellow
set -g message-command-fg blue
set -g message-command-bg black

#window mode
setw -g mode-bg colour6
setw -g mode-fg colour0

# window status
setw -g window-status-format " #F#I:#W#F "
setw -g window-status-current-format " #F#I:#W#F "
setw -g window-status-format "#[fg=magenta]#[bg=black] #I #[bg=cyan]#[fg=colour8] #W "
```

```
setw -g window-status-current-format "#[bg=brightmagenta]#[fg=colour8] #I #[fg=colour8]#[bg=colour14] #W
"
setw -g window-status-current-bg colour0
setw -g window-status-current-fg colour11
setw -g window-status-current-attr dim
setw -g window-status-bg green
setw -g window-status-fg black
setw -g window-status-attr reverse

# Info on left (I don't have a session display for now)
set -g status-left ''

# loud or quiet?
set-option -g visual-activity off
set-option -g visual-bell off
set-option -g visual-silence off
set-window-option -g monitor-activity off
set-option -g bell-action none

set -g default-terminal "screen-256color"

# The modes {
setw -g clock-mode-colour colour135
setw -g mode-attr bold
setw -g mode-fg colour196
setw -g mode-bg colour238

# }
# The panes {

set -g pane-border-bg colour235
set -g pane-border-fg colour238
set -g pane-active-border-bg colour236
set -g pane-active-border-fg colour51

# }
# The statusbar {

set -g status-position bottom
set -g status-bg colour234
set -g status-fg colour137
set -g status-attr dim
set -g status-left ''
set -g status-right '#[fg=colour233,bg=colour241,bold] %d/%m #[fg=colour233,bg=colour245,bold] %H:%M:%S '
set -g status-right-length 50
set -g status-left-length 20

setw -g window-status-current-fg colour81
setw -g window-status-current-bg colour238
setw -g window-status-current-attr bold
setw -g window-status-current-format ' #I#[fg=colour250]:#[fg=colour255]#W#[fg=colour50]#F '

setw -g window-status-fg colour138
setw -g window-status-bg colour235
setw -g window-status-attr none
setw -g window-status-format ' #I#[fg=colour237]:#[fg=colour250]#W#[fg=colour244]#F '

setw -g window-status-bell-attr bold
setw -g window-status-bell-fg colour255
setw -g window-status-bell-bg colour1

# }
# The messages {

set -g message-attr bold
set -g message-fg colour232
set -g message-bg colour166

# }
</code>

A Word of Caution
```

I took care of explaining and documenting all suggested changes in this post to make it easy for you to understand what they do and to decide if this is something you want for your tmux.conf as well.

If you do your amount of research on the web you will find plenty of heavily customized tmux configurations. It's really tempting to just copy those and call it a day. Please, for your own sake, don't do that. Trust me, I've been there. I can only recommend that you start out with a plain tmux configuration and add modifications one by one. This is the only way to ensure that you are fully conscious of all the changes you have made. And this is the only way you will actually learn something about tmux along the way. That said: go ahead and look up config files others have published. Take these as source of inspiration and choose wisely what you want to take for your own config.

There is some really interesting stuff out there. I've seen people doing pure magic with their status bars, displaying all kinds of system information that might be interesting. To apply those changes you'll most likely exceed the simple one-liners I've presented in this post. Often there are special scripts involved that need to be loaded, accompanied by multiple lines of configuration changes. I don't want to discourage you from using these. Just be aware that this is a more advanced topic where you can certainly screw some stuff up. Make sure to backup your config accordingly.

Further resources

As I've already told you, there are plenty of resources out there where you can find people presenting their tmux.confs in a similar fashion to what I've done here. Feel free to browse and search for inspiration. Personally I love reading other people's blog posts about their tmux configs.

GitHub is also a great source. Simply search for "tmux.conf" or repos called "dotfiles" to find a vast amount of configurations that are out there.

If you're especially looking for theming options, I can also recommend having a look at /r/unixporn (SFW, in spite of its title). It's a great place where people showcase their fine-tuned and heavily themed unix environments. Some stuff is really nice, some other stuff is only pretty but mostly dysfunctional. From time to time you find people sharing their tmux.conf as well, you can also deliberately search for the term "tmux.conf" to find what you're looking for.

You can also find my complete tmux.conf (along with other configuration files I'm using on my systems) on my personal dotfiles repo on GitHub.

---

https://hackernoon.com/customizing-tmux-b3d2a5050207

The Definitive Guide to Customizing the Tmux Status Line

I assume you already have a .tmux.conf file set up that is already being used. If not first look up setting up .tmux.conf or something similar.

Step 1: How to Change the Status Line

```
set -g status-left <options>

set -g status-right <options>
```

Replace <options> with a string of all the options you want. Background color can be set using #[bg=colour*] where * is some number 0–255 for terminals with 256 color support. To change the foreground color (text color) it’s just #[fg=colour*] (Again replacing * with a number between 0 and 255).

To view all 255 colors at once see the comment of [https://superuser.com/a/1104214] this stack overflow answer:

<code>
for i in {0..255}; do printf "\x1b[38;5;${i}mcolor%-5i\x1b[0m" $i ; if ! ((($i + 1) % 8)); then echo ; fi ; done
</code>

One of the first things I wanted to do was to add the date and time in the status line. Simple enough right. Here is a complete list of options [15]https://linux.die.net/man/3/strftime or for a shorter more readable list see part of my .tmux.conf https://github.com/sbernheim4/dotfiles/blob/master/.tmux.conf#L59.

Adding %H:%M in the <options> string will show the current time in a 24 hour format without seconds (something like 15:38).

Step 2: Tmux Specific Values

Tmux has its own set of values that stand for different things. For example #H refers to the hostname of the computer you are currently on and #S to the index of the current session. For the complete list of options go [17]here and search for this phrase “Character pair Replaced with”

Step 3: Using Bash Commands to Display Cool Stuff

The last key step is being able to display cool information using bash commands. This can include things like battery percentage, current song playing etc. To do this you first have to find a bash script or command that gives you the output you want. For example, I display the current battery percentage using the following command:

```
pmset -g batt | grep [0-9][0-9]% | awk 'NR==1{print $3}' | cut -c 1-3
```

Note: This only works for Macs since pmset is a mac specific terminal command.

I have a file stored on my computer called battery.sh. The file is two lines long. Line 1 is the bash shebang (#!/bin/bash) and line 2 is the command above. To show the battery in the status line, in my .tmux.conf in <options> I have

```
#(path/to/battery.sh)
```

(Pro Tip: adding an emoji like: ♥ before the # will display it as ♥ 64% in the status line which looks cooler than just 64%)

Step 4: Plugins

Alright so these aren’t really plugins so much as they are scripts that other people have already written. Check out [18]https://github.com/jdxcode/tmux-spotify-info for getting the current playing Spotify song, this again is mac only because the developer wrote this script in apple script. But the point is you can find scripts that other people already wrote for your environment and easily add them to your status line.

Step 5: Customizing the Windows

Hopefully you already know the difference between sessions, windows, and panes. if not, quickly read up on that.

Point is, beyond the left and right side of the status line you can also customize how the windows are displayed in the status with:

```
set -g window-status-format <options>
```

where <options> is the same configuration as it was way above. You also can customize the display for the current window you are on with:

```
set -g window-status-current-format <options>
```

See [https://linux.die.net/man/1/tmux] this site again and search “Character pair Replaced with” to view
some specially reserved Tmux commands for windows.

You can also then customize the separators that are used to show the
difference between windows. Customizing the separators may require a patched font to display certain characters. A quick google search of patched fonts should get you some good results. For cool unicode character separators (still might require a patched font even though they’re unicode) check out [http://www.amp-what.com/unicode/search/] this site.

```
hint: try searching stuff like left triangle
```

---

https://zanshin.net/2013/09/05/my-tmux-configuration/

My Tmux Configuration
September 05, 2013

tmux is a terminal multiplexer. It allows you to have multiple virtual consoles open in a single terminal window. Moreover, you can detatch and attach to a tmux session without ending it, allowing you great flexibility when working on remote servers or machines. In many respects it is just like GNU Screen, but is distributed under a BSD license.

Working from tmux: Productive Mouse Free Development from Pragmatic Press I created my initial .tmux.conf file. Over time I have modified my configuration, most recently to improve the status bar information displayed. What follows is a somewhat annotated listing of my tmux configuration.

<code>
```
# use UTF8
set -g utf8
set-window-option -g utf8 on

# make tmux display things in 256 colors
set -g default-terminal "screen-256color"

# set scrollback history to 10000 (10k)
set -g history-limit 10000

# set Ctrl-a as the default prefix key combination
# and unbind C-b to free it up
set -g prefix C-a
unbind C-b

# use send-prefix to pass C-a through to application
bind C-a send-prefix

# shorten command delay
set -sg escape-time 1

# set window and pane index to 1 (0 by default)
set-option -g base-index 1
setw -g pane-base-index 1

# reload ~/.tmux.conf using PREFIX r
bind r source-file ~/.tmux.conf \; display "Reloaded!"

# use PREFIX | to split window horizontally and PREFIX - to split vertically
bind | split-window -h
bind - split-window -v

# Make the current window the first window
bind T swap-window -t 1

# map Vi movement keys as pane movement keys
bind h select-pane -L
bind j select-pane -D
bind k select-pane -U
bind l select-pane -R

# and use C-h and C-l to cycle thru panes
bind -r C-h select-window -t :-
bind -r C-l select-window -t :+

# resize panes using PREFIX H, J, K, L
bind H resize-pane -L 5
bind J resize-pane -D 5
bind K resize-pane -U 5
bind L resize-pane -R 5

# explicitly disable mouse control
setw -g mode-mouse off
set -g mouse-select-pane off
set -g mouse-resize-pane off
set -g mouse-select-window off

# ----------------------
# Copy & Paste
# ----------------------
# provide access to the clipboard for pbpaste, pbcopy
set-option -g default-command "reattach-to-user-namespace -l zsh"
set-window-option -g automatic-rename on

# use vim keybindings in copy mode
```
</code>

```
    setw -g mode-keys vi

    # setup 'v' to begin selection as in Vim
    bind-key -t vi-copy v begin-selection
    bind-key -t vi-copy y copy-pipe "reattach-to-user-namespace pbcopy"

    # update default binding of 'Enter' to also use copy-pipe
    unbind -t vi-copy Enter
    bind-key -t vi-copy Enter copy-pipe "reattach-to-user-namespace pbcopy"

    bind y run 'tmux save-buffer - | reattach-to-user-namespace pbcopy '
    bind C-y run 'tmux save-buffer - | reattach-to-user-namespace pbcopy '

    # ----------------------
    # set some pretty colors
    # ----------------------
    # set pane colors - hilight the active pane
    set-option -g pane-border-fg colour235 #base02
    set-option -g pane-active-border-fg colour240 #base01

    # colorize messages in the command line
    set-option -g message-bg black #base02
    set-option -g message-fg brightred #orange

    # ----------------------
    # Status Bar
    # -----------------------
    set-option -g status on                # turn the status bar on
    set -g status-utf8 on                  # set utf-8 for the status bar
    set -g status-interval 5               # set update frequencey (default 15 seconds)
    set -g status-justify centre           # center window list for clarity
    # set-option -g status-position top    # position the status bar at top of screen

    # visual notification of activity in other windows
    setw -g monitor-activity on
    set -g visual-activity on

    # set color for status bar
    set-option -g status-bg colour235 #base02
    set-option -g status-fg yellow #yellow
    set-option -g status-attr dim

    # set window list colors - red for active and cyan for inactive
    set-window-option -g window-status-fg brightblue #base0
    set-window-option -g window-status-bg colour236
    set-window-option -g window-status-attr dim

    set-window-option -g window-status-current-fg brightred #orange
    set-window-option -g window-status-current-bg colour236
    set-window-option -g window-status-current-attr bright

    # show host name and IP address on left side of status bar
    set -g status-left-length 70
    set -g status-left "#[fg=green]: #h : #[fg=brightblue]#(curl icanhazip.com) #[fg=yellow]#(ifconfig
en0 | \
    grep 'inet ' | awk '{print \"en0 \" $2}') #(ifconfig en1 | grep 'inet ' | awk '{print \"en1 \" $2}')
\
    #[fg=red]#(ifconfig tun0 | grep 'inet ' | awk '{print \"vpn \" $2}') "

    # show session name, window & pane number, date and time on right side of
    # status bar
    set -g status-right-length 60
    set -g status-right "#[fg=blue]#S #I:#P #[fg=yellow]:: %d %b %Y #[fg=green]:: %l:%M %p :: #(date -u |
\
    awk'{print $4}')::"
</code>
```

The copy and paste section comes from [19]Tmux Copy & Paster on OS X: A Better Future.

For a time I used tmux-powerline for my status bar. However this felt like a sledgehammer for peanuts kind of solution. This past weekend I removed tmux-powerline and created the status bar setup shown above. With the potential to show four IP address the left side of the status bar is a bit

crowded, but I only ever use tmux in a full screen terminal window so I have real estate to spare.

The right side of the status bar displays the session name, screen and pane numbers, date, and local time, and finally UTC time. The middle of the status bar shows a list of the current windows.

The status bar colors are loosely based on tmux-colors-solarized.

You can see my .tmux.conf, along with my entire dotfiles collection by visiting my [23]dotfiles repository on GitHub.
Author's profile picture

---
http://vtechify.com/customizing-tmux-conf/

Customizing your Tmux .conf file

If found a great little write-up on something I had been meaning to jot down for some time. It's all about tmux and how to really take control of your experience while using it.

Tmux is great. I really enjoy using it when spending time in the terminal. There are a ton of things you can change in your config file to really make it yours when using tmux.

Check out Hermann Vocke's [14]post on making tmux pretty and usable.

Using Hermann's post, I've made my own tmux config changes.

Normal Stuff
The most common stuff to change in your tmux.conf file is the prefix key. The default for this is C-b which is odd to type every time you need to send the prefix. I prefer mine to be C-z.

```
# remap prefix from 'C-b' to 'C-z'
unbind C-b
set-option -g prefix C-z
bind-key C-z send-prefix
```

Splitsies
For some reason, the default split screen shortcuts are crazy. Who wants to type C-z " to split vertically and C-z % for horizontal? What? That's dumb. There's a simpler way.

```
# split panes using | and -
bind | split-window -h
bind - split-window -v
unbind '"'
unbind %
```

Reload the config
Might as well make it easy to reload your tmux config anytime.

```
bind r source-file ~/.tmux.conf
```

Fast pane-switch
Changing the view is a frequent thing you do once you start working in tmux. Why not make it super-fast?

I'm not quite fond of triggering the prefix key all the time. I want to be able to simply say M- to go where I want to go (remember: M is for Meta, which is usually your Alt key). With this modification I can simply press Alt-left to go to the left pane (and other directions respectively)

```
# switch panes using Alt-arrow without prefix
bind -n M-Left select-pane -L
bind -n M-Right select-pane -R
bind -n M-Up select-pane -U
bind -n M-Down select-pane -D
```

Make it beautiful
If you're going to spend time there, it might as well look great.

I really like the color scheme Hermann uses. tmux beauty

```
<code>
######################
### DESIGN CHANGES ###
```

```
######################
# panes
set -g pane-border-fg black
set -g pane-active-border-fg brightred
## Status bar design
# status line
set -g status-utf8 on
set -g status-justify left
set -g status-bg default
set -g status-fg colour12
set -g status-interval 2
# messaging
set -g message-fg black
set -g message-bg yellow
set -g message-command-fg blue
set -g message-command-bg black
#window mode
setw -g mode-bg colour6
setw -g mode-fg colour0
# window status
setw -g window-status-format " #F#I:#W#F "
setw -g window-status-current-format " #F#I:#W#F "
setw -g window-status-format "#[fg=magenta]#[bg=black] #I #[bg=cyan]#[fg=colour8] #W "
setw -g window-status-current-format "#[bg=brightmagenta]#[fg=colour8] #I #[fg=colour8]#[bg=colour14] #W
"
setw -g window-status-current-bg colour0
setw -g window-status-current-fg colour11
setw -g window-status-current-attr dim
setw -g window-status-bg green
setw -g window-status-fg black
setw -g window-status-attr reverse
# Info on left (I don't have a session display for now)
set -g status-left ''
# loud or quiet?
set-option -g visual-activity off
set-option -g visual-bell off
set-option -g visual-silence off
set-window-option -g monitor-activity off
set-option -g bell-action none
set -g default-terminal "screen-256color"
# The modes {
setw -g clock-mode-colour colour135
setw -g mode-attr bold
setw -g mode-fg colour196
setw -g mode-bg colour238
# }
# The panes {
set -g pane-border-bg colour235
set -g pane-border-fg colour238
set -g pane-active-border-bg colour236
set -g pane-active-border-fg colour51
# }
# The statusbar {
set -g status-position bottom
set -g status-bg colour234
set -g status-fg colour137
set -g status-attr dim
set -g status-left ''
set -g status-right '#[fg=colour233,bg=colour241,bold] %d/%m #[fg=colour233,bg=colour245,bold] %H:%M:%S '
set -g status-right-length 50
set -g status-left-length 20
setw -g window-status-current-fg colour81
setw -g window-status-current-bg colour238
setw -g window-status-current-attr bold
setw -g window-status-current-format ' #I#[fg=colour250]:#[fg=colour255]#W#[fg=colour50]#F '
setw -g window-status-fg colour138
setw -g window-status-bg colour235
setw -g window-status-attr none
setw -g window-status-format ' #I#[fg=colour237]:#[fg=colour250]#W#[fg=colour244]#F '
setw -g window-status-bell-attr bold
setw -g window-status-bell-fg colour255
setw -g window-status-bell-bg colour1
```

```
# }
# The messages {
set -g message-attr bold
set -g message-fg colour232
set -g message-bg colour166
# }
</code>
```

---

http://blog.hawkhost.com/2010/06/28/tmux-the-terminal-multiplexer/

TMUX – The Terminal Multiplexer (Part 1)

tmux is similar to screen as it lets you run numerous TTY's in the same terminal window. It supports some very cool and intuitive features natively as well as a much more readable configuration syntax (ever looked at a .screenrc file?).

Why TMUX over Screen?

Well according to the FAQ for tmux it has the following advantages over screen:

* A clearly defined cilent/server model (windows are their own clients which allows flexibility on how you handle windows. You can attach and detach different windows in different sessions without any issues)
* Consistent, well-documented command interface. (You can use the same commands interactively as in the .tmux.conf file, more on that later)
* Easily scriptable
* Multiple paste buffers
* Vi & Emacs keybindings
* A more usable status line syntax (which also allows you to embed the output of a shell command, handy indeed.

Default keybindings & Functionality

The default keybindings for tmux are actually pretty intuitive, though if you're used to screen you'll likely get a little peeved with the default action binding of C-b, though this is easily changed to mimic screens behavior:

NOTE If you're like me the Ctrl-b binding isn't horribly intuitive especially if you're used to screen. You can rebind this by putting the following in ~/.tmux.conf:

```
set -g prefix Ctrl-a
```

* Ctrl-b c Create new window
* Ctrl-b d Detach current client
* Ctrl-b l Move to previously selected window
* Ctrl-b n Move to the next window
* Ctrl-b p Move to the previous window
* Ctrl-b & Kill the current window
* Ctrl-b , Rename the current window
* Ctrl-b % Split the current window into two panes
* Ctrl-b q Show pane numbers (used to switch between panes)
* Ctrl-b o Switch to the next pane
* Ctrl-b ? List all keybindings

Now these are pretty self explanatory – the real magic (for me) of tmux is the ease of modifying the default behavior to do what you want, but first things first: let's explore the default behavior of tmux.

Basic Window Handling

Start up tmux with the tmux command and you should be greeted with a simplistic terminal window that resembles screen – the only difference is it has a default status bar which is nice (easily added to screen as well).

Now let's create a couple of windows and go through them (we'll be using the default bindings). Hit Ctrl-b c a few times to create a few windows, you should notice that there are more tabs in the status bar.

Now if you're like me you like to have descriptive names of which each window is for, so let's rename them by hitting Ctrl-b ,. It should prompt you to rename the current window – type anything you want and hit enter. Now the current window is renamed to what you specified. Now going forward I'm going to have two windows open respectively named "window1" and "window2".

Once you rename your windows lets switch back and forth. We have several different ways of switching windows, so I'll go over the ones I personally use:

* Ctrl-b n (Move to the next window)
* Ctrl-b p (Move to the previous window)
* Ctrl-b l (Move to the previously selected window)
* Ctrl-b w (List all windows / window numbers)
* Ctrl-b <window number> (Move to the specified window number, the default bindings are from 0 – 9)

Now these ones fairly self explanatory however they don't really cater to a lot of different windows. What if you have 10+ windows open? It becomes quite tedious to find the window you want – but don't fret! Tmux has a find-window option & keybinding. Type Ctrl-b f and type in the window name you want (it actually searches for the window so you can type in only part of the name of the window you're looking for).

You can also get a list of the windows in the current session by executing the list-windows command. To execute commands interactively you type Ctrl-b : which will bring up a text prompt. From there you can execute any command tmux supports interactively (tab completion is supported).

Basic Pane Handling

One of the most powerful features tmux offers is the ability to split up your current window into "panes". Anyone whose familiar with tiling windows managers will feel quite at home. It's a bit difficult to explain this in words so a simple screenshot will suffice:

Now here are some basic key bindings and commands to split the terminal window (vertically and horizontally) and to switch between them

* Ctrl-b % (Split the window vertically)
* Ctrl-b : "split-window" (Split window horizontally)
* Ctrl-b o (Goto next pane)
* Ctrl-b q (Show pane numbers, when the numbers show up type the key to goto that pane)
* Ctrl-b { (Move the current pane left)
* Ctrl-b } (Move the current pane right)

Now some obviously the default bindings don't encompass some of features, such as splitting horizontally. I personally rebind the keys so "|" splits the current window vertically, and "-" splits it horizontally. Not the easiest things to type, though easy to remember.

You can achieve this by putting the following in ~/.tmux.conf or by typing it in the interactive prompt (Ctrl-b :). Keep in mind if you do the latter it will only be in effect for that session:

```
unbind %
bind | split-window -h
bind – split-window -v
```

Advanced Window Handling

Now that we went over the basics lets dive a little deeper into some "advanced" features of tmux. This includes moving windows around, linking windows together, switching windows from different sessions and much more. By default tmux doesn't have key bindings for these features, so we'll be entering them in the interactive dialog (accessed by typing Ctrl-b :) – keep in mind tmux is very scriptable and you can easily create your own key bindings for all of these.

Moving Windows

Now if you want to move a window you can use the move-window command. The command to do this:

```
move-window [ –d] [ –s src-window] [ –t dst-window]
```

```
swap-window [ -d] [ -s src-window] [ -t dst-window]
```

Similar to the above command except both windows have to exist – if they both do the window with the ID source and destination windows will be swapped.

Advanced Pane Handling

When you split up a window into multiple smaller windows they're referred to as panes. Tmux also offers "layouts" for the panes, or the default positioning and behavior when you create a new window. You can switch through the panes by using the key binding Ctrl-b <space> which will toggle through the different layouts. Each one has different behaviors such as main-vertical which means the current active pane will take up more space in the current window, or even-vertical which will split the panes equally. Since this is difficult to describe in text I believe a few screen shots are in order:

Now that you've seen the different layouts let's see what we can do with these panes. As mentioned above in the "Pane Handling" section you can switch through panes by issuing the Ctrl-b o key combination (which is using the down-pane command) or by typing Ctrl-b q which will list the pane ID's and you select the one you want.

Make your pane into its own window
   If you want to take a pane and make it into its own window you do the following:

     Ctrl-b : “break-pane”

   Simple enough, you should now have the pane in its brand new window. If you don’t want it to automatically make the pane you just broke out as the active window issue the “-d” switch which will simply break the pane to a new window but keep you in the current window.

Resizing Panes
   You can also resize panes if you don’t like the layout defaults. I personally rarely need to do this, though it’s handy to know how. Here is the basic syntax to resize panes:

     Ctrl-b : resize-pane (By default it resizes the current pane down)
     Ctrl-b : resize-pane -U (Resizes the current pane upward)
     Ctrl-b : resize-pane -L (Resizes the current pane left)
     Ctrl-b : resize-pane -R (Resizes the current pane right)
     Ctrl-b : resize-pane 20 (Resizes the current pane down by 20 cells)
     Ctrl-b : resize-pane -U 20 (Resizes the current pane upward by 20 cells)
     Ctrl-b : resize-pane -L 20 (Resizes the current pane left by 20 cells)
     Ctrl-b : resize-pane -R 20 (Resizes the current pane right by 20 cells)
     Ctrl-b : resize-pane -t 2 20 (Resizes the pane with the id of 2 down by 20 cells)
     Ctrl-b : resize-pane -t -L 20 (Resizes the pane with the id of 2 left by 20 cells)
     … etc

   Hopefully you get the jist – don’t get confused! Simply load up a tmux session and split the window a couple of times and issue the above commands. It should become fairly evident how it behaves after fiddling with it for a bit.

Utilizing the client / server model
   I’ve avoided mentioning that a lot of these commands can actually be applied to numerous tmux sessions which allows quite a bit of flexibility – the reason for avoiding it is it’s too much information all at once! An example of using this functionality is if you have two sessions open you can “link” or “move” windows across different sessions – unfortunately the actual “how-to” will be in Part 2.

Conclusion
   Tmux may be a bit confusing however it’s worth putting in a few minutes to check it out and see what it has to offer – quick easy and intuitive.

---
https://wiki.archlinux.org/index.php/tmux

tmux

   tmux is a "terminal multiplexer: it enables a number of terminals (or windows), each running a separate program, to be created, accessed, and controlled from a single screen. tmux may be detached from a screen and continue running in the background, then later reattached."

   tmux is an ISC-licensed alternative to GNU Screen. Although similar, there are many differences between the programs, as noted on the tmux FAQ page.

Installation
   Install the tmux package. Optionally, install [58]tmux-bash-completion^AUR to provide bash completion functions for tmux.

Configuration
   A user-specific configuration file should be located at ~/.tmux.conf, while a global configuration file should be located at /etc/tmux.conf.

Key bindings
   By default, command key bindings are prefixed by Ctrl-b. For example, to vertically split a window type Ctrl-b+%.

   After splitting a window into multiple panes, a pane can be resized by the hitting prefix key (e.g. Ctrl-b) and, while continuing to hold Ctrl, press Left/Right/Up/Down. Swapping panes is achieved in the same manner, but by hitting o instead of a directional key.

   Key bindings may be changed with the bind and unbind commands in tmux.conf. For example, the default prefix binding of Ctrl-b can be changed to Ctrl-a by adding the following commands in your

configuration file:

```
unbind C-b
set -g prefix C-a
bind C-a send-prefix
```

Tip: Quote special characters to use them as prefix. You may also use Alt (called Meta) instead of Ctrl. For example: set -g prefix m-'\'

Additional ways to move between windows include the following:

```
Ctrl-b l (Move to the previously selected window)
Ctrl-b w (List all windows / window numbers)
Ctrl-b <window number> (Move to the specified window number, the default bindings are from 0 – 9)
Ctrl-b q  (Show pane numbers, when the numbers show up type the key to goto that pane)
```

tmux has a find-window option & key binding to ease navigation of many windows:

```
Ctrl-b f <window name> (Search for window name)
Ctrl-b w (Select from interactive list of windows)
```

Copy Mode

A tmux window may be in one of several modes. The default permits direct access to the terminal attached to the window; the other is copy mode. Once in copy mode you can navigate the buffer including scrolling the history. Use vi or emacs-style key bindings in copy mode. The default is emacs, unless VISUAL or EDITOR contains ‘vi’

To enter copy mode do the following:

```
Ctrl-b [
```

You can navigate the buffer as you would in your default editor.

To quit copy mode, use one of the following keybindings:

vi mode:

```
q
```

emacs mode:

```
Esc
```

Browsing URLs

To browse URLs inside tmux you must have [59]urlview^AUR installed and configured.

Inside a new terminal:

```
bind-key u capture-pane \; save-buffer /tmp/tmux-buffer \; run-shell "$TERMINAL -e urlview /tmp/tmux-buffer"
```

Or inside a new tmux window (no new terminal needed):

```
bind-key u capture-pane \; save-buffer /tmp/tmux-buffer \; new-window -n "urlview" '$SHELL -c "urlview < /tmp/tmux-buffer"'
```

Setting the correct term

256 colors

If you are using a 256 colour terminal, you will need to set the correct term in tmux. As of [60]tmux 2.1, this is now tmux, or tmux-256color. You can do this in tmux.conf:

```
set -g default-terminal "tmux-256color"
```

Other, older alternatives, include screen, or screen-256color:

```
set -g default-terminal "screen-256color"
```

Also, if tmux messes up, you can force tmux to assume that the terminal support 256 colors, by adding this in your .bashrc:

```
alias tmux="tmux -2"
```

24-bit color

tmux supports 24-bit color as of version 2.2. If your terminal supports 24-bit color (see this gist), add your terminal to the terminal-overrides setting. For example, if you use Termite, you would add:

```
set -ga terminal-overrides ",xterm-termite:Tc"
```

For other terminals, replace xterm-termite with the relevant terminal type (stored in $TERM). See the tmux(1) man page for details about the Tc terminfo extension.

xterm-keys

To enable xterm-keys in your tmux.conf, you have to add the following line

```
set-option -g xterm-keys on
```

If you enable xterm-keys in your tmux.conf, then you need to build a custom terminfo to declare the new escape codes or applications will not know about them. Compile the following with tic and you can use "xterm-screen-256color" as your TERM:

```
# A screen- based TERMINFO that declares the escape sequences
# enabled by the tmux config "set-window-option -g xterm-keys".
#
# Prefix the name with xterm- since some applications inspect
# the TERM *name* in addition to the terminal capabilities advertised.
xterm-screen-256color|GNU Screen with 256 colors bce and tmux xterm-keys,

# As of Nov'11, the below keys are picked up by
# .../tmux/blob/master/trunk/xterm-keys.c:
        kDC=\E[3;2~, kEND=\E[1;2F, kHOM=\E[1;2H,
        kIC=\E[2;2~, kLFT=\E[1;2D, kNXT=\E[6;2~, kPRV=\E[5;2~,
        kRIT=\E[1;2C,

# Change this to screen-256color if the terminal you run tmux in
# doesn't support bce:
        use=screen-256color-bce,
```

Other Settings

To limit the scrollback buffer to 10000 lines:

```
set -g history-limit 10000
```

Terminal emulator settings can be overridden with

```
set -ga terminal-overrides ',xterm*:smcup@:rmcup@'
set -ga terminal-override ',rxvt-uni*:XT:Ms=\E]52;%p1%s;%p2%s\007'
```

Mouse can be toggled with

```
bind-key m set-option -g mouse on \; display 'Mouse: ON'
bind-key M set-option -g mouse off \; display 'Mouse: OFF'
```

Autostart with systemd

There are some notable advantages to starting a tmux server at startup. Notably, when you start a new tmux session, having the service already running reduces any delays in the startup.

Furthermore, any customization attached to your tmux session will be retained and your tmux session can be made to persist even if you have never logged in, if you have some reason to do that (like a heavily scripted tmux configuration or shared user tmux sessions).

The service below starts tmux for the specified user (i.e. start with tmux@username.service):

```
/etc/systemd/system/tmux@.service
[Unit]
Description=Start tmux in detached session

[Service]
Type=forking
User=%I
ExecStart=/usr/bin/tmux new-session -s %u -d
ExecStop=/usr/bin/tmux kill-session -t %u

[Install]
WantedBy=multi-user.target
```

Tip: You may want to add WorkingDirectory=custom_path to customize working directory.

Alternatively, you can place this file within your [64]systemd/User directory (without User=%I), for example ~/.config/systemd/user/tmux.service. This way the tmux service will start when you log in, unless you also enable [65]systemd/User#Automatic start-up of systemd user instances.

Session initialization

You can have tmux open a session with preloaded windows by including those details in your ~/.tmux.conf:

```
new  -n WindowName Command
neww -n WindowName Command
neww -n WindowName Command
```

To start a session with split windows (multiple panes), include the splitw command below the neww you

would like to split; thus:

```
new  -s SessionName -n WindowName Command
neww -n foo/bar foo
splitw -v -p 50 -t 0 bar
selectw -t 1
selectp -t 0
```

would open 2 windows, the second of which would be named foo/bar and would be split vertically in half (50%) with foo running above bar. Focus would be in window 2 (foo/bar), top pane (foo). Note: Numbering for sessions, windows and panes starts at zero, unless you have specified a base-index of 1 in your .conf

To manage multiple sessions, source separate session files from your conf file:

```
# initialize sessions
bind F source-file ~/.tmux/foo
bind B source-file ~/.tmux/bar
```

X clipboard integration

Tip: The tmux plugin [66]tmux-yank provides similar functionality.

It is possible to copy tmux selection to X clipboard (and to X primary/secondary selection) and in reverse direction. The following tmux config file snippet effectively integrates X clipboard/selection with the current tmux selection using the program [67]xsel:

```
# Emacs style
bind-key -T copy-mode y send-keys -X copy-pipe-and-cancel "xsel -i -p && xsel -o -p | xsel -i -b"
bind-key C-y run "xsel -o | tmux load-buffer - ; tmux paste-buffer"

# Vim style
bind-key -T copy-mode-vi y send-keys -X copy-pipe-and-cancel "xsel -i -p && xsel -o -p | xsel -i -b"
bind-key p run "xsel -o | tmux load-buffer - ; tmux paste-buffer"
```

xclip could also be used for that purpose, unlike xsel it works better on printing raw bitstream that doesn't fit the current locale. Nevertheless, it is neater to use xsel instead of xclip, because xclip does not close STDOUT after it has read from tmux's buffer. As such, tmux doesn't know that the copy task has completed, and continues to wait for xclip's termination, thereby rendering tmux unresponsive. A workaround is to redirect STDOUT of xclip to /dev/null, like in the following:

```
# Vim style
bind-key -T copy-mode-vi y send-keys -X copy-pipe-and-cancel "xclip -i -sel clip > /dev/null"
bind-key p run "xclip -o -sel clip | tmux load-buffer - ; tmux paste-buffer"
```

Urxvt middle click

Note: To use this, you need to enable mouse support

There is an unofficial perl extension ([69]mentioned in the official [70]FAQ) to enable copying/pasting in and out of urxvt with tmux via Middle Mouse Clicking.

First, you will need to download the perl script and place it into urxvts perl lib:

```
wget [71]http://anti.teamidiot.de/static/nei/*/Code/urxvt/osc-xterm-clipboard
mv osc-xterm-clipboard /usr/lib/urxvt/perl/
```

You will also need to enable that perl script in your .Xdefaults:

```
~/.Xdefaults
...
*URxvt.perl-ext-common:         osc-xterm-clipboard
...
```

Next, you want to tell tmux about the new function and enable mouse support (if you haven't already):

```
~/.tmux.conf
...
set-option -ga terminal-override ',rxvt-uni*:XT:Ms=\E]52;%p1%s;%p2%s\007'
set -g mouse on
...
```

That's it. Be sure to end all instances of tmux before trying the new MiddleClick functionality.

While in tmux, Shift+MiddleMouseClick will paste the clipboard selection while just MiddleMouseClick will paste your tmux buffer. Outside of tmux, just use MiddleMouseClick to paste your tmux buffer and your standard Ctrl-c to copy.

Tips and tricks

Start tmux with default session layout

Session managers like tmuxinator and tmuxp make it easy to manage common session configurations.

For tmuxinator, install tmuxinator^AUR from AUR. Test your installation with

```
tmuxinator doctor
```

Get the default layout values

Start tmux as usual and configure your windows and panes layout as you like. When finished, get the current layout values by executing (while you are still within the current tmux session)

```
tmux list-windows
```

The output may look like this (two windows with 3 panes and 2 panes layout)

```
0: default* (3 panes) [274x83] [layout
20a0,274x83,0,0{137x83,0,0,3,136x83,138,0[136x41,138,0,5,136x41,138,42,6]}] \
@2 (active)
1: remote- (2 panes) [274x83] [layout e3d3,274x83,0,0[274x41,0,0,4,274x41,0,42,7]] @3
```

The Interesting part you need to copy for later use begins after [layout... and excludes ...] @2 (active). For the first window layout you need to copy e.g. 20a0,274x83,0,0{137x83,0,0,3,136x83,138,0[136x41,138,0,5,136x41,138,42,6]}

Define the default tmux layout

Knowing this, you can exit the current tmux session. Following this, you create your default tmux session layout by editing tmuxinator's config file (Don't copy the example, get your layout values as described above)

~/.tmuxinator/default.yml

```
name: default
root: ~/
windows:
  - default:
      layout: 20a0,274x83,0,0{137x83,0,0,3,136x83,138,0[136x41,138,0,5,136x41,138,42,6]}
      panes:
        - clear
        - vim
        - clear && emacs -nw
  - remote:
      layout: 24ab,274x83,0,0{137x83,0,0,3,136x83,138,0,4}
      panes:
        -
        -
```

The example defines two windows named "default" and "remote". With your determined layout values. For each pane you have to use at least one - line. Within the first window panes you start the commandline "clear" in pane one, "vim" in pane two and "clear && emacs -nw" executes two commands in pane three on each tmux start. The second window layout has two panes without defining any start commmands.

Test the new default layout with (yes, it is "mux"):

```
mux default
```

Autostart tmux with default tmux layout

If you like to start your terminal session with your default tmux session layout edit ~/.bashrc

```
 if [ -z "$TMUX" ]; then
   mux default
 fi
```

Alternate approach for default session

Instead of using the above method, one can just write a bash script that when run, will create the default session and attach to it. Then you can execute it from a terminal to get the pre-designed configuration in that terminal

```
#!/bin/bash
tmux new-session -d -n WindowName Command
tmux new-window -n NewWindowName
tmux split-window -v
tmux selectp -t 1
tmux split-window -h
tmux selectw -t 1
tmux -2 attach-session -d
```

Start tmux in urxvt

Use this command to start urxvt with a started tmux session. I use this with the exec command from my .ratpoisonrc file.

```
urxvt -e bash -c "tmux -q has-session && exec tmux attach-session -d || exec tmux new-session -n$USER -
s$USER@
$HOSTNAME"
```

Start tmux on every shell login

Bash

For bash, simply add the following line of bash code to your .bashrc before your aliases; the code for other shells is very similar:

~/.bashrc

```
# If not running interactively, do not do anything
[[ $- != *i* ]] && return
[[ -z "$TMUX" ]] && exec tmux
```

Note: This snippet ensures that tmux is not launched inside of itself (something tmux usually already checks for anyway). tmux sets $TMUX to the socket it is using whenever it runs, so if $TMUX isn't set or is length 0, we know we aren't already running tmux.

Add the following snippet to start only one session (unless you start some manually), on login, try attach at first, only create a session if no tmux is running.

```
# TMUX
if which tmux >/dev/null 2>&1; then
    #if not inside a tmux session, and if no session is started, start a new session
    test -z "$TMUX" && (tmux attach || tmux new-session)
fi
```

The following snippet does the same thing, but also checks tmux is installed before trying to launch it. It also tries to reattach you to an existing tmux session at logout, so that you can shut down every tmux session quickly from the same terminal at logout.

```
# TMUX
if which tmux >/dev/null 2>&1; then
    # if no session is started, start a new session
    test -z ${TMUX} && tmux

    # when quitting tmux, try to attach
    while test -z ${TMUX}; do
        tmux attach || break
    done
fi
```

Another possibility is to try to attach to existing detached session or start a new session:

```
if [[ -z "$TMUX" ]] ;then
    ID="$( tmux ls | grep -vm1 attached | cut -d: -f1 )" # get the id of a deattached session
    if [[ -z "$ID" ]] ;then # if not available create a new one
        tmux new-session
    else
        tmux attach-session -t "$ID" # if available attach to it
    fi
fi
```

Start a non-login shell

tmux starts a login shell by default, which may result in multiple negative side effects:

* Users of fortune may notice that quotes are printed when creating a new panel.
* The configuration files for login shells such as ~/.profile are interpreted each time a new panel is created, so commands intended to be run on session initialization (e.g. setting audio level) are executed.

To disable this behaviour, add to ~/.tmux.conf:

```
set -g default-command "${SHELL}"
```

Use tmux windows like tabs

The following settings added to ~/.tmux.conf allow to use tmux windows like tabs, such as those provided by the reference of these hotkeys — [78]urxvt's tabbing extensions^[[79]broken link: invalid section]. An advantage thereof is that these virtual “tabs” are independent of the terminal emulator.

```
#urxvt tab like window switching (-n: no prior escape seq)
bind -n S-down new-window
bind -n S-left prev
bind -n S-right next
bind -n C-left swap-window -t -1
```

```
bind -n C-right swap-window -t +1
```

Of course, those should not overlap with other applications' hotkeys, such as the terminal's. Given that they substitute terminal tabbing that might as well be deactivated, though.

It can also come handy to supplement the EOT hotkey Ctrl+d with one for tmux's detach:

```
bind-key -n C-j detach
```

Clients simultaneously interacting with various windows of a session

In Practical Tmux, Brandur Leach writes:

> Screen and tmux's behaviour for when multiple clients are attached to one session differs slightly. In Screen, each client can be connected to the session but view different windows within it, but in tmux, all clients connected to one session must view the same window. This problem can be solved in tmux by spawning two separate sessions and synchronizing the second one to the windows of the first, then pointing a second new session to the first.

The script “tmx” below implements this — the version here is slightly modified to execute “tmux new-window” if “1” is its second parameter. Invoked as tmx <base session name> [1] it launches the base session if necessary. Otherwise a new “client” session linked to the base, optionally add a new window and attach, setting it to kill itself once it turns “zombie”.

```
tmx
```

```
<code>
#!/bin/bash

#
# Modified TMUX start script from:
#     http://forums.gentoo.org/viewtopic-t-836006-start-0.html
#
# Store it to `~/bin/tmx` and issue `chmod +x`.
#

# Works because bash automatically trims by assigning to variables and by
# passing arguments
trim() { echo $1; }

if [[ -z "$1" ]]; then
    echo "Specify session name as the first argument"
    exit
fi

# Only because I often issue `ls` to this script by accident
if [[ "$1" == "ls" ]]; then
    tmux ls
    exit
fi

base_session="$1"
# This actually works without the trim() on all systems except OSX
tmux_nb=$(trim `tmux ls | grep "^$base_session" | wc -l`)
if [[ "$tmux_nb" == "0" ]]; then
    echo "Launching tmux base session $base_session ..."
    tmux new-session -s $base_session
else
    # Make sure we are not already in a tmux session
    if [[ -z "$TMUX" ]]; then
        echo "Launching copy of base session $base_session ..."
        # Session id is date and time to prevent conflict
        session_id=`date +%Y%m%d%H%M%S`
        # Create a new session (without attaching it) and link to base session
        # to share windows
        tmux new-session -d -t $base_session -s $session_id
        if [[ "$2" == "1" ]]; then
                # Create a new window in that session
                tmux new-window
        fi
        # Attach to the new session & kill it once orphaned
        tmux attach-session -t $session_id \; set-option destroy-unattached
    fi
fi
```

</code>

   A useful setting for this is
setw -g aggressive-resize on

   added to ~/.tmux.conf. It causes tmux to resize a window based on the smallest client actually viewing it, not on the smallest one attached to the entire session.

   An alternative is to put the following ~/.bashrc:
.bashrc
```
function rsc() {
  CLIENTID=$1.`date +%S`
  tmux new-session -d -t $1 -s $CLIENTID \; set-option destroy-unattached \; attach-session -t $CLIENTID
}

function mksc() {
  tmux new-session -d -s $1
  rsc $1
}
```

   Citing the author:
          "mksc foo" creates a always detached permanent client named "foo". It also calls "rsc foo" to create a client to newly created session. "rsc foo" creates a new client grouped by "foo" name. It has destroy-unattached turned on so when I leave it, it kills client. Therefore, when my computer looses network connectivity, all "foo.something" clients are killed while "foo" remains. I can then call "rsc foo" to continue work from where I stopped.

Correct the TERM variable according to terminal type
   Instead of setting a fixed TERM variable in tmux, it is possible to set the proper TERM (either screen or screen-256color) according to the type of your terminal emulator:
~/.tmux.conf
```
## set the default TERM
set -g default-terminal screen

## update the TERM variable of terminal emulator when creating a new session or attaching a existing session
set -g update-environment 'DISPLAY SSH_ASKPASS SSH_AGENT_PID SSH_CONNECTION WINDOWID XAUTHORITY TERM'
## determine if we should enable 256-colour support
if "[[ ${TERM} =~ 256color || ${TERM} == fbterm ]]" 'set -g default-terminal screen-256color'
```

~/.zshrc
```
## workaround for handling TERM variable in multiple tmux sessions properly from
[83]http://sourceforge.net/p/
tmux/mailman/message/32751663/ by Nicholas Marriott
if [[ -n ${TMUX} && -n ${commands[tmux]} ]];then
        case $(tmux showenv TERM 2>/dev/null) in
                *256color) ;&
                TERM=fbterm)
                        TERM=screen-256color ;;
                *)
                        TERM=screen
        esac
fi
```

Reload an updated configuration without restarting tmux
   By default tmux reads ~/.tmux.conf only if it was not already running. To have tmux load a configuration file afterwards, execute:
tmux source-file <path>

   This can be added to ~/.tmux.conf as e. g.:
bind r source-file <path>

   You can also do ^: and type :
source .tmux.conf

Template script to run program in new session resp. attach to existing one
   This script checks for a program presumed to have been started by a previous run of itself. Unless found it creates a new tmux session and attaches to a window named after and running the program. If however the program was found it merely attaches to the session and selects the window.
```
#!/bin/bash

PID=$(pidof $1)
```

```
if [ -z "$PID" ]; then
    tmux new-session -d -s main ;
    tmux new-window -t main -n $1 "$*" ;
fi
    tmux attach-session -d -t main ;
    tmux select-window -t $1 ;
exit 0
```

A derived version to run irssi with the nicklist plugin can be found on its ArchWiki page.

Terminal emulator window titles

If you SSH into a host in a tmux window, you'll notice the window title of your terminal emulator remains to be user@localhost rather than user@server. To allow the title bar to adapt to whatever host you connect to, set the following in ~/.tmux.conf

```
set -g set-titles on
set -g set-titles-string "#T"
```

For set-titles-string, #T will display user@host:~ and change accordingly as you connect to different hosts.

Automatic layouting

When creating new splits or destroying older ones the currently selected layout isn't applied. To fix that, add following binds which will apply the currently selected layout to new or remaining panes:

```
bind-key -n M-c kill-pane \; select-layout
bind-key -n M-n split-window \; select-layout
```

Vim friendly configuration

With tmux 2.4 change:

```
bind -t vi-copy 'v' begin-selection
bind -t vi-copy 'y' copy-selection
bind -t vi-copy 'Space' halfpage-down
bind -t vi-copy 'Bspace' halfpage-up
```

to:

```
bind-key -T copy-mode-vi 'v' send -X begin-selection
bind-key -T copy-mode-vi 'y' send -X copy-selection
bind-key -T copy-mode-vi 'Space' send -X halfpage-down
bind-key -T copy-mode-vi 'Bspace' send -X halfpage-up
```

Troubleshooting

Scrolling issues

If you have issues scrolling with Shift-Page Up/Down in your terminal, the following will remove the smcup and rmcup capabilities for any term that reports itself as anything beginning with xterm:

```
set -ga terminal-overrides ',xterm*:smcup@:rmcup@'
```

This tricks the terminal emulator into thinking tmux is a full screen application like pico or mutt[, which will make the scrollback be recorded properly. Beware however, it will get a bit messed up when switching between windows/panes. Consider using tmux's native scrollback instead.

Mouse scrolling

Note: This interferes with selection buffer copying and pasting. To copy/paste to/from the selection buffer hold the shift key.

If you want to scroll with your mouse wheel, ensure mode-mouse is on in .tmux.conf

```
set -g mouse on
```

You can set scroll History with:

```
set -g history-limit 30000
```

For mouse wheel scrolling as from tmux 2.1 try adding one or both of these to ~/.tmux.conf

```
bind -T root WheelUpPane   if-shell -F -t = "#{alternate_on}" "send-keys -M" "select-pane -t =; copy-mode \
-e; send-keys -M"
bind -T root WheelDownPane if-shell -F -t = "#{alternate_on}" "send-keys -M" "select-pane -t =; send-keys -M"
```

Though the above will only scroll one line at a time, add this solution to scroll an entire page instead

```
bind -t vi-copy    WheelUpPane   page-up
```

```
   bind -t vi-copy    WheelDownPane page-down
   bind -t emacs-copy WheelUpPane   page-up
   bind -t emacs-copy WheelDownPane page-down
```

Terminal emulator does not support UTF-8 mouse events

   When the terminal emulator does not support the UTF-8 mouse events and the mouse on tmux option is set, left-clicking inside the terminal window might paste strings like [M# or [Ma into the promt.

   To solve this issue set:

```
set -g mouse-utf8 off
```

Shift+F6 not working in Midnight Commander

   See Midnight Commander#Broken shortcuts.

---

https://sanctum.geek.nz/arabesque/tmux-environment-variables/

Tmux environment variables

   The user configuration file for the tmux terminal multiplexer, .tmux.conf, supports defining and using environment variables in the configuration, with the same syntax as most shell script languages:

```
TERM=screen-256color
set-option -g default-terminal $TERM
```

   This can be useful for any case in which it may be desirable to customise the shell environment when inside tmux, beyond setting variables like default-terminal. However, if you repeat yourself in places in your configuration file, it can also be handy to use them as named constants. An example could be establishing colour schemes:

```
TMUX_COLOUR_BORDER="colour237"
TMUX_COLOUR_ACTIVE="colour231"
TMUX_COLOUR_INACTIVE="colour16"

set-window-option -g window-status-activity-bg $TMUX_COLOUR_BORDER
set-window-option -g window-status-activity-fg $TMUX_COLOUR_ACTIVE
set-window-option -g window-status-current-format "#[fg=$TMUX_COLOUR_ACTIVE]#I:#W#F"
set-window-option -g window-status-format "#[fg=$TMUX_COLOUR_INACTIVE]#I:#W#F"
```

   The explicit commands to work with environment variables in .tmux.conf are update-environment, set-environment, and show-environment, and are featured in the [http://www.openbsd.org/cgi-bin/man.cgi?query=tmux] manual.
