tmux workflow

龔存

April 20, 2020

Contents

1 基礎知識

1.1 安裝 tmux

tmux 依賴 libevent 庫, 先編譯安裝:

```
# uncompress the .tar.gz file
gtar xfz libevent-2.0.21-stable.tar.gz
cd libevent-2.0.21-stable

# install the gnu m4 and autoconf tools
yum install -y m4
ln -sf /opt/freeware/bin/m4 /usr/bin/

yum whatprovides autoconf autoheader
yum install -y automake autoconf

# change AC_AUTOCONF_VERSION to the current version
aclocal --version
vi aclocal.m4

# install
autoheader -I.
autoconf
./configure

gmake -j4
gmake install
```

在 AIX 編譯安裝 tmux 有點麻煩, 要手動設置使用 ncurses 庫 (否則出現 term.h 無法識別的警告並影響色彩的顯示):

```
cd tmux-1.9a-source
autoheader
autoconf
./configure --prefix=/usr/local \
            LDFLAGS="-L/usr/local/lib -L/opt/freeware/lib" \
            CPPFLAGS="-I/opt/freeware/include/ncurses" LIBS="-lncurses" CC=xlc
gmake -j4
gmake install
```

1.2 開始使用 tmux

在終端中只要執行以下命令就可以啟動:

```
$ tmux
```

要關閉會話, 只要輸入:

```
$ exit
```

但是, 除非只是很短暫地使用 tmux, 否則不建議這樣使用會話. **必須**使用" 命名會話" (named session) 來取代這種方式.

1.2.1 創建命名會話

我們可以在一台計算機上創建一個叫做 basic 的會話:

```
$ tmux new-session -s basic
```

這個命令可以簡化為:

```
$ tmux new -s basic
```

命名會話便於我們管理 tmux 的後台運行會話.

1.2.2 分離和連接會話

tmux 的最大優勢是啟動 tmux 環境並執行各種程序或進程時, 可以通過從會話" 分離" (detaching) 讓 tmux 在後台運行.

如果我們關閉了一個普通的終端會話 (正常或收到 SIGHUP 信號而掛斷), 那麼這個會話中的所有進程都會被殺死 (ignore SIGHUP 除外). 但是從一個 tmux 會話分離時, 實際並沒有關閉 tmux. 在這個會話中運行的程序仍然在運行. 然後可以在任何想要的時候再" 連接" (attaching) 到這個會話, 你會發現所有的界面就和分離會話時一模一樣. 下面演示這個功能, 我們首先創建一個命名會話, 啟動 topas 程序, 然後從這個會話中分離:

```
$ tmux new -s basic
```

然後, 在這個 tmux 會話中執行 topas 命令, 它在 AIX 中檢測系統內存和 CPU 使用情況:

```
$ topas
```

現在按下 PREFIX 鍵 (命令前綴, 默認為 **Ctrl-b**), 然後再按下 **d** 鍵, 這樣就從 tmux 會話中分離了, 返回到標準終端界面里了.

1.2.3 命令前綴

由於我們的程序是在 tmux 環境裡運行的, 因此需要一種方式告訴 tmux 當前所輸入的命令, 而不是終端的輸入, 這就是 PREFIX 的作用. 當我們想要從 tmux 會話中分離時, 可以先按 PREFIX 鍵 (由於命令前綴可以定制, 因此一律用 PREFIX 代替), 再按 d 鍵 (d=detach).

1.2.4 重新連接已有會話

我們之前新建了一個 tmux 會話, 在會話中運行了一個程序, 然後從這個會話中分離出來, 最後甚至關閉終端窗口, 但是這個 tmux 會話依然在運行, 而 topas 命令也沒有停止運行.

在一個新的終端中執行以下命令列出當前存在的 tmux 會話:

```
$ tmux list-sessions
```

可以簡化為:

```
$ tmux ls
```

這個命令會展示目前有一個會話正在運行:

```
basic: 1 windows (created Wed Apr 15 21:32:09 2020) [132x39]
```

想要連接這個會話, 可以使用 attach 命令

```
$ tmux attach -t basic
```

这個命令可以簡化為

```
$ tmux a -t basic
```

这个会话可以被重命名

```
$ tmux rename -t basic new-name
```

1.2.5 杀死会话

可以在一个会话中使用 exit 来退出并杀死会话, 也可以使用 kill-session 命令杀死指定会话:

```
$ tmux kill-session -t basic
```

此时如果再次列出当前已有会话, 会得到如下信息:

```
$ tmux ls
failed to connect to server: Connection refused
```

这是因为目前没有 tmux 会话在运行, 因此 tmux 本身也没有运行.

1.3 使用窗口

在一个 tmux 会话中同时运行多个命令或同时执行多个程序的情景非常普遍. 可以通過窗口 (windows) 來管理它們.

一個新的 tmux 窗口被創建時, tmux 環境會自動創建一個初始化的窗口. 我們可以創建多個窗口, 並且分離會話之後它們會一直存在.

現在我們創建一個包含 2 個窗口的新會話. 第一個窗口是 shell, 第二個窗口將執行 topas 命令. 可以用以下命令創建一個叫做 term 的會話:

```
$ tmux new -s term -n shell
```

我們已經為第一個窗口命名為 shell, 按下 **PREFIX c** 創建第二個窗口, tmux 會自動把焦點切換到這個新的窗口. 在這裡可以運行其他程序, 在這個窗口里我們運行 **topas** 命令. 但是第二個窗口並沒有命名 (默認為 bash), 我們通過 **PREFIX ,** (PREFIX+comma) 來進行命令窗口 (類似于 emacs 中的 minibuffer).

可以在一個 tmux 會話中創建任意多個窗口. 但是一旦創建了兩個以上窗口, 就必須學會在窗口之間切換 (move).

有多種方法可以在這些窗口之間來回切換. 切換到上一個窗口 (preview-window) 默認是'PREFIX p' 鍵, 切換到下一個窗口 (next-window) 默認是'PREFIX n'[1].

tmux 的窗口有默認編號 (從 0 開始計數). 按下 PREFIX 0 切換到第 1 個窗口, 按下 PREFIX 1 切換到第 2 個窗口. 如果窗口超過 9 個, 可以按下 PREFIX f 鍵來查找窗口 (如果窗口已被命名), 或者按下 PREFIX w (w=windows) 顯示一個可視化的窗口列表, 然後再選擇其中想要的那個窗口.

如果想要關閉一個窗口, 可以按下 PREFIX & (&=et=exit). 要想完全退出一個 tmux 會話, 必須要關閉所有窗口.

[1]為了不和之後配置衝突, 已改為 **PREFIX M-p**

1.4 使用面板

tmux 可以將窗口再分割成多個面板. 在我的定制中, 水平分割面板使用 PREFIX -, 垂直分割面板使用 PREFIX |, 在面板之間來回切換使用 PREFIX o, 還可以通過 PREFIX 前綴, 後面跟隨 h, j, k, l 來上下左右切換 (同 vim 按鍵). 按 PREFIX SPC 鍵 (空格) 可以依次選取面板佈局. 關閉面板可以使用 PREFIX x 或者 PREFIX q.

1.5 使用命令模式

到現在為止, 我們都是使用 tmux 組合鍵來操作, 我們也可以通過 tmux 的 minibuffer (command area) 來執行命令. 通過執行 **PREFIX :** (冒號鍵) 進入命令模式, 比如創建一個新的窗口并命名為 console:

```
new-window -n console
```

再進一步, 創建一個窗口并執行 topas 命令:

```
new-window -n cmd "topas"
```

在創建 tmux 窗口時給它一個初始化的進程是非常便捷的, 但是按下 **q** 關閉 topas 時, 這個 tmux 窗口也會被一起關閉. 但可以通過配置文件來使得其不關閉窗口.

Table 1: Commands for Sessions, Windows, and Panes (Details see ~/.tmux.conf)

Command	Description
tmux new-session	Creates a new session without a name. Can be shortened to tmux new or simply tmux.
tmux new -s development	Creates a new session called ”development”.
tmux new -s development -n editor	Creates a session named ”development” and names the first window ”editor”.
tmux attach -t development	Attaches to a session named ”development”.
tmux list-keys	List key bindings.
PREFIX d	Detaches from the session, leaving the session running in the background.
PREFIX :	Enters Command mode.
PREFIX c	Creates a new window from within an existing tmux session. Shortcut for new-window.
PREFIX 0..9	Selects windows by number.
PREFIX w	Displays a selectable list of windows in the current session.
PREFIX ,	Displays a prompt to rename a window.
PREFIX &	Closes the current window after prompting for confirmation.
PREFIX -	Divides the current window in half horizontally.
PREFIX |	Divides the current window in half vertically.
PREFIX o	Cycles through open panes.
PREFIX SPC	Cycles through the various pane layouts.
PREFIX x	Closes the current pane after prompting for confirmation.
PREFIX C-q	Momentarily displays pane numbers in each pane.

2 配置 tmux

在默認情況下, tmux 會在兩個位置查找配置文件. 首先查找 ~/.tmux.conf 作為系統配置, 然後在當前用戶的主目錄下查找 /etc/tmux.conf 文件. 如果這兩個文件都不存在, tmux 就會使用默認配置. 我們主要通過配置 ~/.tmux.conf 文件來定制 tmux.

2.1 定義更方便的前綴鍵

tmux 默認使用 Ctrl-b 作為 PREFIX, 如果將 CAPS LOCK 映射為 CTRL 之後, PREFIX 設為 Ctrl-a 將更為方便.

在 ~/.tmux.conf 中, 我們使用 **set-option** 命令來設置選項, 可以縮寫為 **set**

```
set -g prefix C-a
```

這裡我們使用 **-g** 選項, 即全局配置. 儘管不是必須的, 我們可以通過 **unbind-key** 命令或 **unbind** 命令移除之前的組合鍵

```
unbind C-b
```

tmux 并不會 **實時**地自動地從配置文件讀取修改. 因此如果你在使用 tmux 的過程中修改了 ~/.tmux.conf 文件, 要讓配置修改生效的話, 需要關閉所有的 tmux 會話然後重新打開它, 要麼在 tmux 中發送一個命令來重新加載配置文件. 現在我們來自定義一個快捷鍵來讓它重新加載配置文件:

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

這樣就綁定了 C-r 來 reload 配置文件, 儘管上面的命令沒有 PREFIX, 但是在使用 bind 定義快捷鍵后, 還是需要在實際中先按下 PREFIX, 再按下 **C-r** 鍵. 雖然我們自定義了加載配置文件的快捷鍵, 但是在新的配置文件被加載前我們還是不能使用它, 因此還需要再使用一次'PREFIX :' 進入命令模式, 然後輸入以下命令重新加載配置文件:

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

重新加載配置文件后, tmux 並不會提示配置是否改變, 最好通過 **display** 命令讓 tmux 在狀態欄輸出一個消息:

```
bind C-r source-file ~/.tmux.conf \; display "Reloaded!"
```

通過在多個命令之間添加 \; 符號可以使一個鍵綁定多個命令. 通過剛才定義的快捷鍵, 我們可以在修改配置文件后按下 PREFIX C-r 鍵使新的配置快速生效.

我們把前綴鍵重新映射到了 C-a 鍵, 但是例如 vim, emacs 甚至是 bash 終端等也會用到這個組合鍵, 我們需要配置 tmux, 把這個組合鍵發送給需要的程序中. 可以定義一個快捷鍵來發送 send-prefix 命令:

```
bind C-a send-prefix
```

在配置生效后, 只需要按兩次 C-a 就可以把 C-a 命令發送給 tmux 里的程序了.

其他鍵定義 (分割面板, 重新映射移動鍵等), 請參考 ~/.tmux.conf 文件.

2.2 視圖風格

為了讓 tmux 具有最佳的視覺體驗, 首先要確保終端和 tmux 都運行在 256 色模式中, 我們首先配置終端.

2.2.1 設置 PuTTY

- PuTTY configuration → Window → Colours → Allow terminal to use xterm 256-colour mode
- PuTTY configuration → Connection → Data → Terminal-type string → xterm-256color

2.2.2 定制顏色

我們可以通過以下簡單的腳本來測試和選擇想要的顏色

```
export TERM=xterm-256color
bash -c 'for i in {0..255}; \
 do printf "\x1b[38;5;${i}mcolour${i}\n"; done'
```

各類窗口, 面板, 及裝填欄的顏色請參考 ~/.tmux.conf 文件.

Table 2: For Future Reference

Command	Description
set -g prefix C-a	Sets the key combination for the Prefix key.
set -sg escape-time n	Sets the amount of time (in milliseconds) tmux waits for a keystroke after pressing Prefix.
source-file [file]	Loads a configuration file. Use this to reload the existing configuration or bring in additional configuration options later.
bind C-a send-prefix	Configures tmux to send the prefix when pressing the Prefix combination twice consecutively.
bind-key [key] [command]	Creates a keybinding that executes the specified command. Can be shortened to bind
bind-key -r [key] [command]	Creates a keybinding that is repeatable, meaning you only need to press the Prefix key once, and you can press the assigned key repeatedly afterwards. This is useful for commands where you want to cycle through elements or resize panes. Can be shortened to bind.
unbind-key [key]	Removes a defined keybinding so it can be bound to a different command. Can be shortened to unbind.
display-message or display	Displays the given text in the status message.
set-option [flags] [option] [value]	Sets options for sessions. Using the -g flag sets the option for all sessions.
set-window-option [option] [value]	Sets options for windows, such as activity notifications, cursor movement, or other elements related to windows and panes.
set -a	Appends values onto existing options rather than replacing the option’ s value.

3 腳本定制 tmux 環境

在項目工作時, 可能需要運行一大堆的工具和命令集. 我們可以使用 tmux 的 client-server 模型, 來創建一個定制的腳本來自動地構建開發環境, 分割窗口並運行程序.

3.1 編寫一個項目配置腳本

我們完全可以讓 tmux 創建包含多個窗口, 每個窗口包含多個面板, 並且讓每個面板都運行不同的程序, 使得一鍵執行多個程序, 並且將窗口保持在最舒服的狀態. 以應急協作為例, 我們可以一鍵進入需要運維的服務器, 檢查 topas 狀態, 打開 OMS 菜單隨時準備重啟進程, 並有一個命令行可以輸入任何命令. 為了實現這個能力, 我們準備的腳本如下

```
#!/usr/bin/ksh

# script name: event.tmux
# usage: event.tmux <hostname>
```

```
# result: open two windows, the 1st window will be split to two panes, the 1st
#   pane show the OMS menu, the 2nd pane give you a shell; the 2nd window will
#   execute the 'topas' command.

alias tmux='tmux -2u'
typeset event=event_$$

if tmux has-session -t ${event} 2>/dev/null; then
    echo "session event ${event} existed" >&2
    exit -1
fi

if [ $# -lt 1 ]; then
    echo "miss hostname" >&2
    exit -1
fi
typeset host=$1


tmux new-session -s ${event} -n $host -d
tmux send-keys -t ${event} "ssh -t bunimsvr ssh -t $host" C-m
tmux send-keys -t ${event} "cd /menus" C-m
tmux send-keys -t ${event} "/menus/menu.ksh" C-m

# split windows vertically
tmux split-window -v -t ${event}
tmux send-keys -t $event:0.1 "ssh -t bunimsvr ssh -t $host" C-m

# create a new window to display the topas
tmux new-window -n topas -t $event "ssh -t bunimsvr ssh -t $host topas"

tmux select-window -t ${event}:0
tmux attach -t $event
```

通過執行以上腳本, 將會進入到 tmux 終端畫面, 其中生產系統的操作畫面已經等待著你.

4 和 git 集成

以 **install_scripts** 項目為例, 整個項目的主目錄為 ***smgroup/install_scripts***, 目前只有 **master** 分支, 目前核心維護者是 GongCun. 對於開發者來說, 常規步驟將是:

1. **Fork** the project to create developer's own repository in GitLab.
2. Clone the Git repository with

   ```
   git clone git@bocgitsvr.bocmo.com:{developer}/install_scripts.git \
       ./install_scripts_${developer}
   ```

3. Setup ssh access

   ```
   $ ssh-keygen -f ~/.ssh/${developer}_id_rsa

   ## Copy the content of ${developer}_id_rsa.pub to
   ## 'SSH Keys' in User Settings of gitlab

   $ eval "$(ssh-agent -s)"
   $ ssh-add ~/.ssh/${developer}_id_rsa
   $ ssh -T git@bocgitsvr.bocmo.com
   Welcome to GitLab, ${developer}!
   ```

4. Branch from origin/origin with

```
git fetch origin master
git checkout -b ${developer} origin/master
```

5. Make changes and commit them with 'git add' and 'git commit' in ${developer} branch.
6. Push the new commit to the developer's repository

```
git push -u origin ${developer}
```

7. Create merge request to **master** branch of install_scripts, and ask other team members for review and feedback of the changes.

These steps allow a core maintainer to merge a branch into master branch after successful review:

1. Fetch and check out the branch for this merge request

```
git checkout -D ${developer}/install_scripts-${developer} 2>/dev/null
git fetch git@bocgitsvr.bocmo.com:${developer}/install_scripts.git ${developer}
git checkout -b ${developer}/install_scripts-${developer} FETCH_HEAD
```

   FETCH_HEAD is a short-lived ref, to keep track of what has just been fetched from the remote repository. git pull first invokes git fetch, in normal cases fetching a branch from the remote; FETCH_HEAD points to the tip of this branch (it stores the SHA1 of the commit, just as branches do). git pull then invokes git merge, merging FETCH_HEAD into the current branch.

1. Review the changes locally

```
# compare two branch
git diff master..${developer}/install_scripts-${developer}

# only show the changed files
git diff master..${developer}/install_scripts-${developer} --name-only
```

2. Merge the branch and fix any conflicts that come up

```
git checkout master
git merge --no-ff ${developer}/install_scripts-{developer}
```

3. Push the result of the merge to GitLab

```
git push origin master
```

Tip: You can also checkout merge requests locally by following these guidelines.

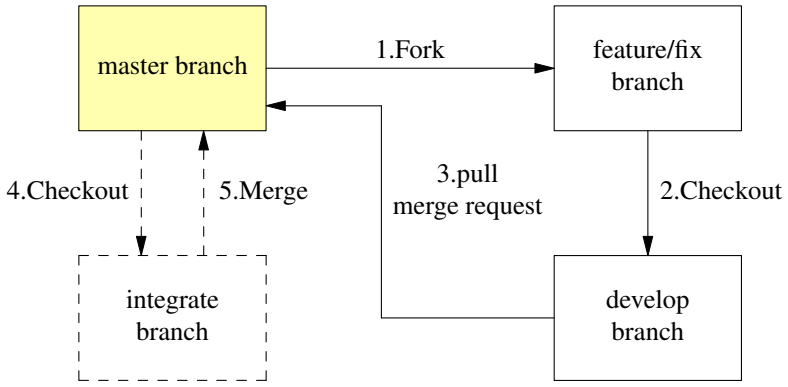

Figure 1: Work flow for developer

4.1 通過 tmux 自動化

可以通過 tmux 來定制 git 開發環境: 我們將用戶從主倉庫 clone 到本地, 並 checkout 到開發分支, 然後通過 vim 打開工作目錄的流程通過 tmux 一氣呵成. 我們通過兩個腳本來實現自動化: **develop-git.sh** 用來執行 git clone 操作, 并調用 **develop.tmux** 腳本來實現打開 vim[2] 并顯示 branch 信息:

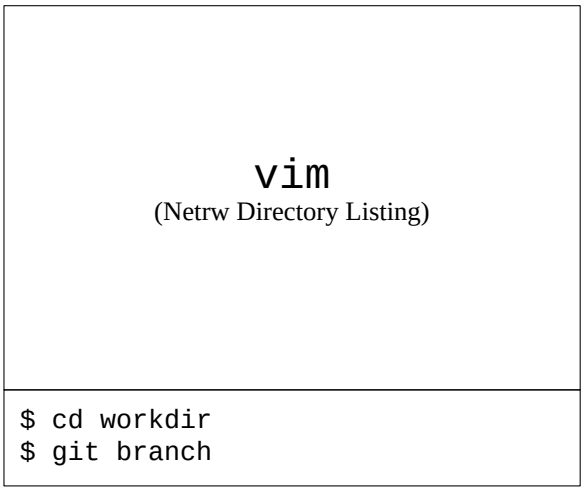

Figure 2: Pnae layout for developing

5 文本和緩衝區

在日常工作中, 使用複製粘貼的次數遠超你的想象. 使用 tmux, 可以通過快捷鍵方便地操作屏幕內容, 比如複製一段面板內容到另一個面板中去, 或者翻看之前的屏幕輸出. 本章將會介紹如何管理 tmux 會話中的文本, 并 DIY 一個插件實現錄屏功能.

5.1 使用複製模式滾動輸出

通過定制 ~/.tmux.conf, 按下'PREFIX ESC' 將進入複製模式 (類似于 vi), 我們在 ~/.tmux.conf 中配置使用了 vi 中的移動鍵, 這樣操作將非常方便:

```
setw -g mode-keys vi
```

這個配置可以使用 h, j, k, l 在緩衝區中移動, 要離開複製模式, 只需按下 ENTER 鍵. 我們也可以使用 w, b, f/F, t/T 等功能鍵輔助移動.

5.1.1 在緩衝區中快速移動

通過 vi 的滾屏鍵可以在緩衝區中快速移動, 這些鍵包括:

- **C-f** 向下翻滾一屏
- **C-b** 向上翻滾一屏
- **g** 跳轉到緩衝區歷史的最頂部
- **G** 跳轉到緩衝區歷史的最底部

[2]通過 vim work-folder 來瀏覽文件. 雖然我使用 emacs, 但是考慮到學習曲線, 仍然建議在項目中使用 vim. vim 8.1 增加了 terminal 功能並且自帶 netrw 插件, 值得花時間掌握.

5.1.2 查找緩衝區

在複製模式中按下 **?** 或 **/** 鍵可以向下或向上查找 char/string, 跳轉到下一個匹配按 **n**, 跳轉到上一個匹配按 **N** (如果按下? 鍵, 則方向相反).

5.1.3 複製和粘貼文本

複製和粘貼文本也非常類似於 vi 操作, 在複製模式中按下 SPACE 鍵將開始選擇文本, 然後可以繼續通過 vi 快捷鍵移動選擇區域, 按下 **y** 鍵將被選擇的區域複製到粘貼緩衝區中[3]. 要粘貼剛才捕獲的內容, 則直接按下 **PREFIX p** 鍵. 由於粘貼緩衝區是一個環形棧, 每複製一個新的文本, 就會把緩存放在棧的最頂端, 可以通過以下鍵綁定來選擇或操作粘貼緩衝區:

- **PREFIX C-s** 顯示粘貼緩衝區內容
- **PREFIX C-l** 列出複製文本清單
- **PREFIX C-c** 選擇要複製的文本 (可以通過 j,k 等鍵移動)
- **PREFIX C-d** 刪除棧頂的文本. 建議使用 PREFIX :[RET] deleteb -b buffer-name 來刪除指定的緩衝區較好.

5.2 如何創建 tmux 插件

由於多重緩衝區的操作已經較為複雜了, 而對於屏幕的記錄, 快照, 及歷史輸出的保存, 已經不能通過簡單的鍵綁定來方便地實現了, 我們可以嘗試自己寫插件.

5.2.1 創建一個新的 git 項目

```
$ mkdir -p ${TMUX_PLUGIN_PATH}/my-logging
$ cd ${TMUX_PLUGIN_PATH}/my-logging
$ git init
```

5.2.2 創建 *.tmux 插件運行文件

```
$ touch my-logging.tmux
$ chmod u+x my-logging.tmux
```

5.2.3 創建關於插件的鍵綁定

我們實現以下鍵綁定功能:

1. **PREFIX M-l** Trigger 終端錄屏 (Meta 是指 Alt 鍵).
2. **PREFIX M-c** 將當前畫面 dump 下來.
3. **PREFIX M-h** 將當前窗口或面板的所有歷史輸出保存下來.

因此 **my-logging.tmux** 的內容如下:

[3]這裡的概念非常類似 emacs. 雖然 tmux 和 vi 模式結合得較好, 但是 tmux 的作者其實是 emacs 用戶, 這是我之前在他的某篇文章中看到的.

```
#!/usr/bin/env bash

CURRENT_DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"
tmux bind-key M-l run-shell "$CURRENT_DIR/scripts/my_logging.sh"
tmux bind-key M-c run-shell "$CURRENT_DIR/scripts/my_capture.sh"
tmux bind-key M-h run-shell "$CURRENT_DIR/scripts/my_history.sh"
```

注意調用的 **.sh** 腳本必須有可執行權限.

5.2.4 實現插件功能

以 **my_logging.sh** 為例:

```
#!/usr/bin/env bash

saved_display_time=$(tmux show-option -gqv display-time)
if [ -z "saved_display_time" ]; then
    saved_display_time=750;
fi

file="/tmp/tmux-log-#S.#W.#P.%Y-%b-%d-%R.log"
tmux set-option -gq display-time 5000
tmux pipe-pane -o "cat >>$file"
tmux display-message "Screen logging to $file"
tmux set-option -gq display-time ${saved_display_time}
```

其實際調用的是 pipe-pane 命令, 并增加了消息顯示等功能.

5.2.5 測試插件

執行以下命令檢查插件是否有效:

```
$ ./my-logging.tmux
```

如果插件有效, 可以在 ~/.tmux.conf 中增加如下配置:

```
run-shell /smgroup/tmux/plugin/my-logging/my-logging.tmux
```

這樣確保 tmux 啟動時自動載入插件.

關於如何編寫插件的更多內容, 可以參考How to create Tmux plugins.

6 使用 tmux 結對編程

tmux 最受人們歡迎的功能之一是結對編程 (pair programming). 遠程用戶協作有兩種方式, 第一種是有一個公用賬戶 (比如 nim 服務器上的 root 用戶), 在這個賬戶下配置 tmux 和工作環境; 第二種方法是使用 tmux 的 sockets 連接, 這樣你就可以讓其他用戶連接到你的 tmux 會話中而無需共享賬戶信息.

6.1 通過共享賬戶結對編程

這是最簡便的方式, A 用戶登錄系統中建立一個會話

```
$ tmux new -s pairing
```

B 用戶可以 attach 到這個會話進行項目協作

```
$ tmux a -t pairing
```

以上方式 A 和 B 通常會看到相同的內容並在同一個窗口中交互, 但很多時候人們希望能夠在獨立的, 不同的窗口工作而不用互相干擾. 使用" 組會話" (grouped session) 可以實現這個功能. 首先 A 用戶創建一個會話:

```
$ tmux new -s pairing
```

B 用戶不是直接 attach 到這個會話, 而是" 創建一個新的會話" 來加入到這個會話中, 但要指定原始會話 pairing, 然後再指定一個 B 用戶自己的會話名:

```
$ tmux new -t pairing -s mysession
```

當第二個會話啟動時, 兩個用戶都可以同時在這個會話里進行交互, 但是每個用戶都可以創建相互獨立的窗口. 第二個用戶可以通過 kill-session 結束自己的會話, 而原始會話仍然存在. 但是, 如果所有的窗口都關閉了, 則原始會話和新會話都會被殺死.

6.2 使用 Socket 結對編程

使用 tmux 提供的 socket 支持, 可以讓多個用戶連接到會話而不用共享賬戶. A 用戶先用自己的賬戶登錄系統, 并通過 socket 創建一個新的 tmux 會話:

```
$ tmux -S /var/tmux/pairing
```

B 用戶只要指定 unix domain socket 路徑并連接, 就可以 attach 會話

```
$ tmux -S /var/tmux/pairing attach
```

使用這種方式時, ~/.tmux.conf 文件是指第一個啟動這個會話的用戶的配置文件. 另外通過 socket 方式啟動的 tmux 會話是不能通過 tmux ls 顯示的, 可以通過 **lsof** 來獲知:

```
$ sudo lsof -U | grep '^tmux'
```