Extracted from:

Remote Pairing

Collaborative Tools for Distributed Development

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Pragmatic Press

Remote Pairing

Collaborative Tools for Distributed Development

Joe Kutner

Edited by Brian P. Hogan

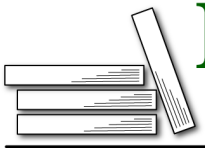
Remote Pairing

Collaborative Tools for Distributed Development

Joe Kutner

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *http://pragprog.com*.

The team that produced this book includes:

Brian P. Hogan (editor)
Candace Cunningham (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Printed in the United States of America.
ISBN-13: 978-1-937785-74-1
Encoded using the finest acid-free high-entropy binary digits.
Book version: P2.0—January 2014

Sharing a tmux Session

To pair-program with tmux, we must connect a second client to our session. Leave the tmux session we created earlier running. Then open a second terminal window (outside of tmux) and run this command:

```
$ tmux attach
```

The second terminal will look identical to the first terminal. That's because it's the same tmux session. Start typing in the first terminal, and we'll see the changes in the second terminal. Likewise, characters entered in the second terminal will appear in the first terminal. If we run a command like `ping`, it will be visible in both consoles. We're only simulating the process of pair programming with tmux in this example. In the real world, the two terminal windows would exist on separate machines, but the result would be the same: a collaborative editing environment.

You may have noticed when connecting a second client to the tmux session that the window is only as big as the smallest terminal, as the following figure shows.

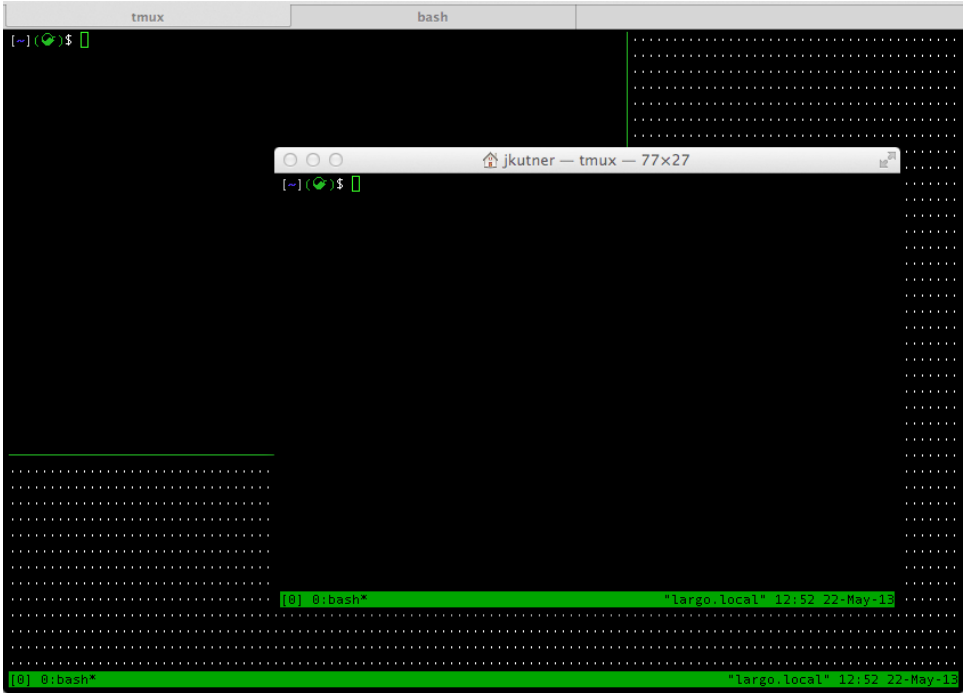


Figure 4—Two tmux windows with different sizes

This is good because it means that both programmers are seeing the exact same thing. But sometimes resizing a terminal that contains a tmux window can cause all sorts of weird display issues. If this happens, just detach the misbehaving client by pressing Ctrl-b d, and reattach with the same command we ran earlier. tmux version 1.8 has reflow support, which largely eliminates this problem.

Now, detach the second session by pressing Ctrl-b and then d in tmux. The second terminal will return to the normal prompt, but the first terminal will remain in the tmux session. Anyone using the same user account as us can rejoin this session with the attach command.

But don't forget that we're going to pair with a programmer we found on Twitter. We don't want that person logging in with our user account. Let's discuss how to securely share this session.

Sharing Sessions Across User Accounts

We could share a tmux session by creating a joint user account that both parties log into, but we would lose all of our dotfiles and any tools we have installed exclusively for our primary user. Instead, we'll create a dedicated account for our pairing partner, and share a tmux session between that account and our primary user account. Some people create a user specifically for each pairing partner, but we'll just create a general tmux user that any partner can log into.

On Mac OS X, create a *tmux* user from the Users & Groups section in the System Preferences application, as Figure 5, *Creating a tmux user on Mac OS X*, on page 7 shows.

On Ubuntu and many other Linux systems, create the tmux user like this:

```
$ adduser tmux
```

Next, we'll give our pairing partner access to this account. Emailing a password is not very safe, so we'll use an SSH key. Switch to the tmux user by running this command:

```
$ su tmux
```

Then create a .ssh and the .ssh/authorized_keys file, setting the appropriate permissions. Only the tmux user should be allowed to read, write, or execute this.

```
$ mkdir ~/.ssh
$ touch ~/.ssh/authorized_keys
$ chmod 700 ~/.ssh
$ chmod 600 ~/.ssh/authorized_keys
```

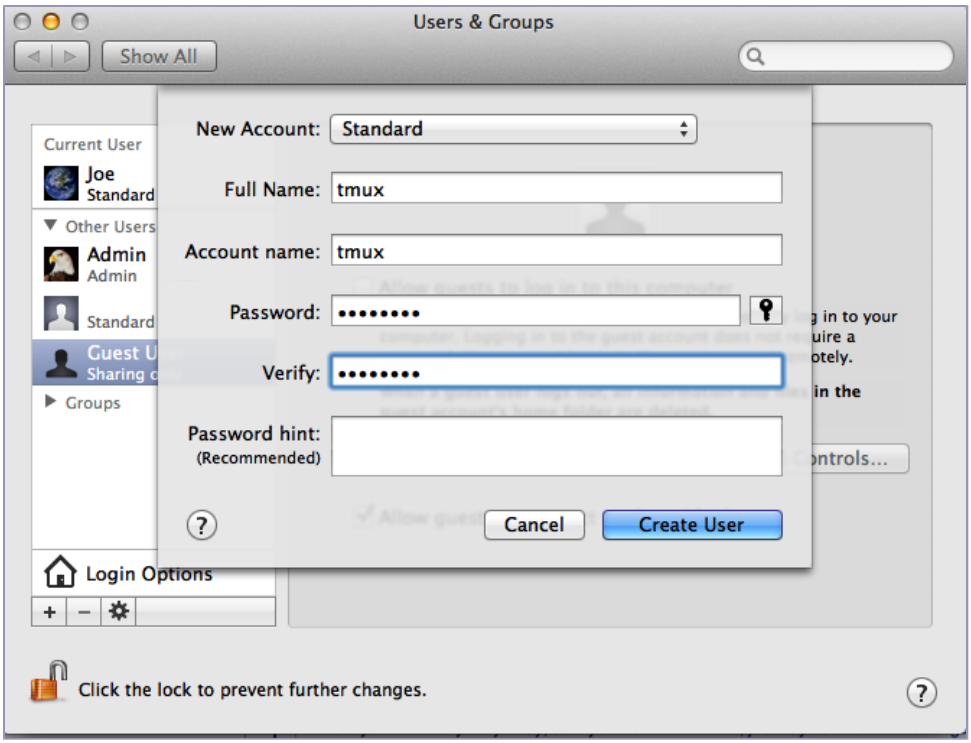


Figure 5—Creating a tmux user on Mac OS X

Next, we'll add our partner's public key to the authorized_keys file. If our partner has a GitHub account, we can use the github-auth Ruby Gem to help us.[6] We installed RubyGems in the *Preface*, on page ?, so we can download and install github-auth with this command:

```
$ gem install github-auth
```

This will add the gh-auth script to our path. We can test it out by adding our own public key like this (replace johndoe with your GitHub username):

```
$ gh-auth add --users=johndoe
Adding 2 key(s) to '/Users/tmux/.ssh/authorized_keys'
```

We could repeat the command with our partner's GitHub username, but we'll skip that step since our partner is imaginary. When we're done with the partner, we can remove her key with the gh-auth remove command.

6. https://github.com/chrishunt/github-auth

We need sudo access for the next steps, but we don't want our guest's user account to have that privilege. Exit the tmux user and return to your user like this:

```
$ exit
```

Now create a *tmux* group by running this command on Linux:

```
$ sudo addgroup tmux
```

On Mac OS X we'll need to create the group from the Users & Groups section in the System Preferences application, as the following figure shows.

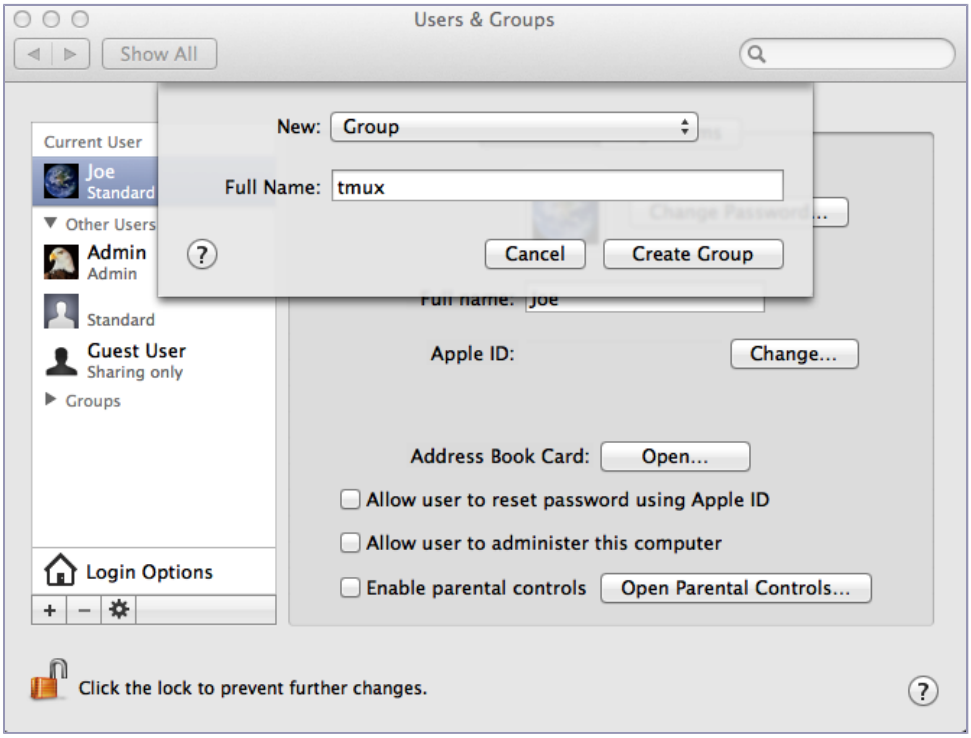


Figure 6—Creating a tmux group on Mac OS X

Now create a /var/tmux directory to hold our shared sessions and change its ownership so the tmux group has access.

```
$ sudo mkdir /var/tmux
$ sudo chgrp tmux /var/tmux
```

Then alter the folder permissions so that new files will be accessible for all members of the tmux group:

```
$ sudo chmod g+ws /var/tmux
```

Finally, add both the tmux user and our user to the tmux group. On Linux, run these commands (replace janedoe with your username):

```
$ sudo usermod -aG tmux tmux
$ sudo usermod -aG tmux janedoe
```

On some systems, including the Vagrant box, we'll have to log out and log back in for these user changes to take effect.

On Mac OS X, we can add the users to the tmux group in Users & Groups, as the following figure shows.

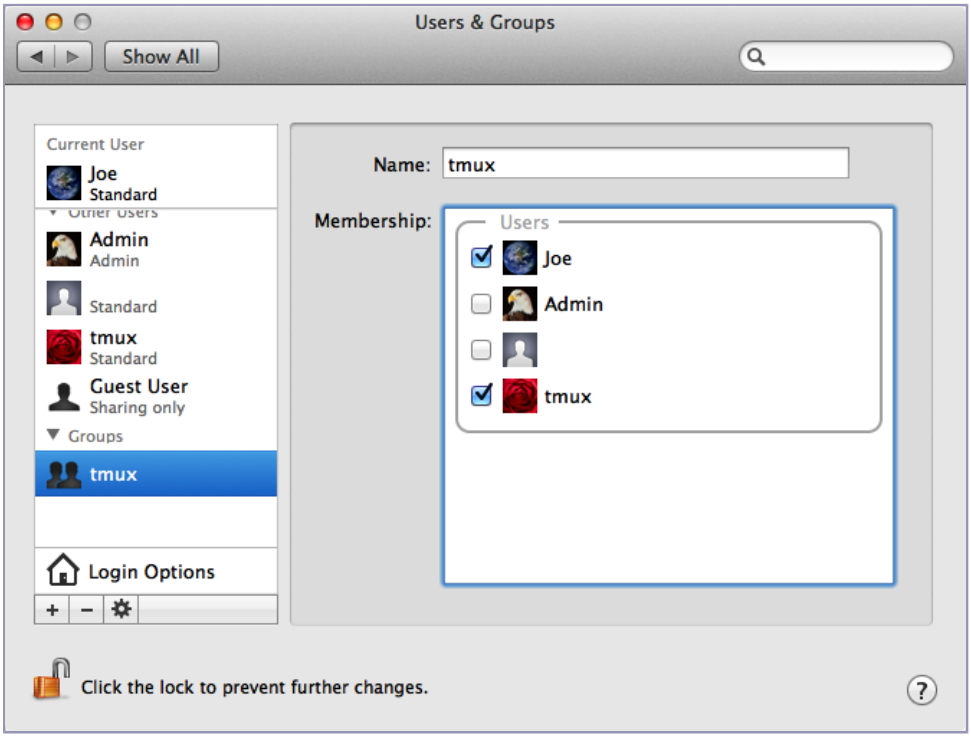


Figure 7—Adding users to the tmux group on Mac OS X

Finally, we must give the tmux user remote login access. On Linux, this is enabled by default. On Mac OS X, open the Sharing section from the System Preferences application and select the Remote Login option from the list of services. Check the box next to it, and select the "All Users" option from the panel on the right-side. If you prefer to give access only to specific users, then you must add the tmux user to the list at the bottom of the panel.

In the next section, we'll create a tmux session that we can share between these two users.

Sharing tmux Sessions with Sockets

If a tmux session is running, terminate it by entering exit in the console. Then run the following command, which will create a pair socket (essentially just a file) under the /var/tmux directory:

```
$ tmux -S /var/tmux/pair
```

The new tmux session will look identical to our old ones, but now a tmux client from another user account can connect to it. Let's do just that.

Open a second terminal, but keep the first terminal open and visible. In the second terminal, log in with the tmux user just as a remote partner would (if you're using Vagrant you'll have to run this from within the virtual machine).

```
$ ssh tmux@localhost
The authenticity of host 'localhost (::1)' can't be established.
RSA key fingerprint is 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'host' (RSA) to the list of known hosts.
Password:
Last login: Fri Mar 29 09:52:08 2013
tmux@localhost$
```

Now run this command, which will connect a second tmux client to the pair session.

```
tmux@localhost$ tmux -S /var/tmux/pair attach
```

The two sessions, which are simulating remote machines, will be connected. There are more-complicated approaches to sharing sessions, such as using a socat tunnel or a reverse proxy.[7] In Chapter 3, *Using the Cloud to Connect*, on page ?, you'll learn how to do the latter, which will help us connect across a wide area network. The approach described here works best when we're sharing across a local area network or using a virtual private network.

Now let's move on to the code.

7. http://thread.gmane.org/gmane.comp.terminal-emulators.tmux.user/579