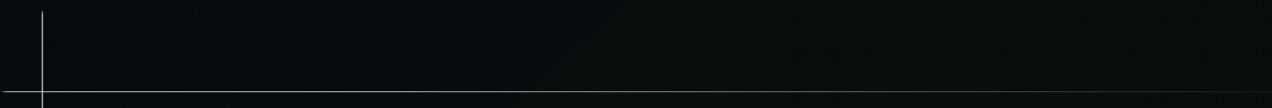
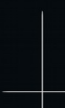


Managing Linux Servers via CSSH

Jeff Smith
aka CrankyOldBugger
crankyoldbugger@gmail.com



Usage

- manage multiple SSH sessions at the same time
 - this tool is intended for managing multiple systems where the same configuration or commands can be run concurrently
 - handy for cluster management as well
 - works on RedHat Enterprise, Redhat Fedora, CentOS, Debian and Ubuntu
 - “should” work on any POSIX compliant UNIX style OS, i.e. cygwin
-
- 

Installation

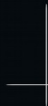
- Ubuntu/Debian/PopOS

```
sudo apt-get install -y clusterssh
```

- Fedora/RHEL/Centos

```
sudo yum (dnf) install -y clusterssh
```

GitHub: <https://github.com/duncs/clusterssh.git>



Caveats

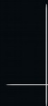
Like many UNIX tools, ClusterSSH has the potential to go horribly awry if you aren't very careful with its use.

It would take 100 men 100 years to screw things up the way CSSH can in one millisecond... seriously...

I mean it...

trust me on this...

Super Fancy Tip: Remember to try to use global commands such as 'sed' instead of just depending on cursor movements, as the file you're editing might not be the same across all clients.



Global Configuration Files

There are three global configuration files:


`/etc/clusters`

- Contains a list of cluster names and the hosts mapped to the specified cluster.

`/etc/tags`

- Reversed logic to `/etc/clusters`. This allows you to specify one host as a member of multiple tags.

`/etc/csshrc`

- This file contains default configuration overrides.
-
- 

User Specific Configuration Files

There are also three user specific files that can overwrite the global files.

`$HOME/.clusterssh/clusters`

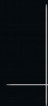
- User specific version of `/etc/clusters`

`$HOME/.clusterssh/tags`

- User specific version of `/etc/tags`

`$HOME/.clusterssh/config`

- User specific version of `/etc/csshrc`
-



The Config Files

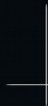
Global Config:

`/etc/sshrc`

Personal Config:

`$HOME/.clusterssh/config`

Default options are overwritten first by the global file, and then by the user file.



An Example Config File

<code>## /etc/csshrc</code>	
<code>screen_reserve_top = 25</code>	Number of pixels from the screen's side to reserve when calculating screen geometry for tiling. Setting this to something like 50 will help keep cssh from positioning windows over your window manager's menu bar if it draws one at that side of the screen.
<code>screen_reserve_bottom = 25</code>	
<code>screen_reserve_left = 20</code>	
<code>screen_reserve_right = 20</code>	
<code>terminal_font = 6x12</code>	Font to use in the terminal windows. Use standard X font notation. You can find available fonts in <code>/usr/share/fonts/X11/misc/font.alias</code>
<code>terminal_reserve_top = 80</code>	Number of pixels from the terminal's side to reserve when calculating screen geometry for tiling. Setting these will help keep cssh from positioning windows over your scroll and title bars or otherwise overlapping the windows too much.
<code>terminal_reserve_bottom = 25</code>	
<code>terminal_reserve_left = 5</code>	
<code>terminal_reserve_right = 5</code>	
<code>terminal_size = 100x90</code>	Initial size of terminals to use. NOTE: the number of lines (24) will be decreased when resizing terminals for tiling, not the number of characters (80).
<code>window_tiling = yes</code>	Perform window tiling (set to 0 to disable)

Running cssh ad-hoc

You can run cssh as-is (without a pre-made client list) using this one-line command:

```
cssh -l username server_ip_address_1 server_ip_address_2  
server_ip_address_3
```

example:

```
cssh -l root fedora01 fedora02 centos01
```

Running with a “clusters” file

You can run `cssh` using a global or local clusters file with:

`cssh -l username clustername` (no username in file)

`cssh clustername` (username specified in file)

example:

`cssh -l root clusterubuntu`

Formats:

`cssh <clustername> -l username <server>[:port] <server>[:port] [...]`



A Sample /etc/clusters

```
# clusters = my ubuntu systems  
clusterubuntu ubuntu01 ubuntu02 ubuntu03
```

```
# clusters = my fedora systems  
clusterfedora fedora01 fedora02 centos01
```

```
# clusters = all of my systems  
clusterall ubuntu01 ubuntu02 ubuntu03 fedora01 fedora02 centos01
```

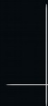
Or:

```
# clusters = my ubuntu systems with user name  
clusterubuntu jeff@ubuntu01 jeff@ubuntu02 jeff@ubuntu03
```

Running with a “tags” file

You can run `cssh` using a global or local tags file with:

```
cssh -l username tagname
```



A Sample \$HOME/.clustersssh/tags

```
#KWLUG test
```

```
ubuntu01  ubuntu kwlug
```

```
ubuntu02  ubuntu kwlug
```

```
ubuntu03  ubuntu kwlug
```

```
fedora01  fedora kwlug
```

```
fedora02  fedora kwlug
```

```
centos01  fedora centos kwlug
```



Nifty Parameters

`--action '<command>', -a '<command>'`

Run the command in each session, e.g. `"-a 'vi /etc/hosts'"` to drop straight into a vi session.

`--fillscreen`

Resize terminal windows to fill the whole available screen

`--port <port>, -p <port>`

Specify an alternate port for connections.

`--tag-file '<filename>', -r '<filename>'`

Use supplied file as additional tag file (see also "FILES")

`--unique-servers, -u`

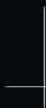
Toggle connecting to each host only once when a hostname has been specified multiple times.

`--username '<username>', -l '<username>'`

Specify the default username to use for connections (if different from the currently logged in user). NOTE: will be overridden by `<user>@<host>`.

See <https://github.com/duncs/clusterssh> for more.

There's more parameters at <https://linux.die.net/man/1/cssh> as well



PSSH (parallel ssh)

Similar to cssh, but generally only does one command per session.

Parameters:

-h host_file

--hosts host_file

Read hosts from the given host_file. Lines in the host file are of the form [user@]host[:port] and can include blank lines and comments (lines beginning with "#"). If multiple host files are given (the -h option is used more than once), then pssh behaves as though these files were concatenated together. If a host is specified multiple times, then pssh will connect the given number of times.

-l user

--user user

Use the given username as the default for any host entries that don't specifically specify a user.

Examples:

Connect to host1 and host2, and print "hello, world" from each:

```
pssh -i -H "host1 host2" echo "hello, world"
```

Print "hello, world" from each host specified in the file hosts.txt:

```
pssh -i -h hosts.txt echo "hello, world"
```