08 – Your shell and working remotely

CS 2043: Unix Tools and Scripting, Spring 2019 [1]

Matthew Milano

February 6, 2019

Cornell University

Table of Contents

As always: Everybody! ssh to wash.cs.cornell.edu

- Quiz time! Everybody! run `quiz-02-08-19`
- You can just explain a concept from last class, doesn't have to be a command this time.

More on shell customization

Aliases

Creating Aliases

`alias <new-name> <old-name>`

- Aliases **`new-name`** to be **`old-name`**, e.g. `alias ..='cd ..'`
  - Can now type `..` to go up one directory.
- Should not ever be used in scripts.
  - Disabled by default, battle to use them — **very** bad practice.
  - I don't have your aliases, so now I can't run your script.
- Usually stored in **`~/.<shell>rc`** file, though **`~/.<shell>_aliases`** is slowly gaining traction.
  - Make sure you **`source ~/.<shell>_aliases`** from **`~/.<shell>rc`** or else they won't be available!!!
  - E.g. **`bash`**: **`~/.bashrc`** sources **`~/.bash_aliases`**, or
  - **`zsh`**: **`~/.zshrc`** sources **`~/.zsh_aliases`**

Modifying your Terminal Prompt

- The `$PS1` variable controls what shows up when you type in your terminal.
  - In `zsh` this is `$PROMPT`.
- List of all options here.
- Common: `export PS1="\u@\h:\w> "`
  - `usr@hostname:current/working/directory>`
- Try changing your `$PS1` using `export` right now to see how you can modify it.
- Play with colors after, since they are tedious to type in the format needed.

Storing Customizations

- There are many such places that people put things, but generally speaking...
- Your `bashrc` should have things like aliases and functions. Limit the `export` calls to just things related to coloring the terminal.
- Your `bash_profile` should contain any special environment variables you need to define.
    - Typically when you are exporting things like `$PATH` or `$LD_LIBRARY_PATH` for something you have installed on your own.
- You should source your `bash_profile` from your `profile`, and you should source your `bashrc` from your `bash_profile`.

Working Remotely

Some Terminology

- The server you are logging into is called the `remote` (host).
- The user (you) are referred to as the `client`.
- If you obtain access to a *cluster* (many individual nodes presented together), you may encounter terms such as:
  - The `head` node (sometimes called `master`).
  - The `worker` nodes (sometimes called the `slaves`).
    - While `master` and `slave` are common terms, we prefer (and encourage adoption of) the terms `head` and `worker`.
  - You often are only allowed to log into the `head` node directly.
  - There is usually a queueing system (e.g., `qsub`) that submits `jobs` that get farmed out to the `workers`.
  - In most scenarios, you get charged by the number of cores / resources being used.

`ssh` Examples

- On `ugclinux` (CS Undergraduate servers) I am `mpm288`:
  - v1: `ssh mpm288@ugclinux.cs.cornell.edu`
  - v2: `ssh -l mpm288 ugclinux.cs.cornell.edu`
- Sweet! `ugclinux` has Matlab, can I use it?

```
$ /usr/local/MATLAB/R2012a/bin/matlab
Warning: No display specified. You will not be able to
         display graphics on the screen.
>> exit()
# exit() left Matlab
$ exit # close the ssh connection
```

- Now do: `ssh -X mpm288@ugclinux.cs.cornell.edu`

```
$ /usr/local/MATLAB/R2012a/bin/matlab
# Matlab displays on my screen now!
```

CS Servers: More Information

- More info: https://it.cornell.edu/coecis/linux-ugc-lab-computing-and-information-science-cis

Important Excerpt from Above Article

Students should copy or delete their files in home directories at the end of each academic year. Home directories for students not currently enrolled in a CS course will be purged to reclaim server storage space. If you need assistance copying files off the server, please submit a Help Desk ticket.

Transferring Files

Secure Copy

`scp [flags] <from> <to>`

- It's exactly like **`cp`**, only you are transferring over the web.
- Can transfer *from* the **`client`** *to* the **`remote`** host.
- Can transfer *from* the **`remote`** host *to* the **`client`**.
- Copy directories just like before using the **`-r`** flag.
- Must specify the **`user`** on the **`remote`** host.
- **`Remote`** syntax (for **`<from>`** component):
  `user@host.name:/path/to/file/or/folder`
  - You need the **`:`** to start the **`path`**.
- If you don't have permission...you can't get it!
- More modern systems may even let you **`TAB`** complete across the **`remote`** directories :)

`scp` Examples

- Transfer from `remote` to local computer:

```
$ scp mpm288@blargh.ru:/home/mpm288/colorize.sh ~/Desktop/
colorize.sh                                   100% 3299     3.2KB/s   00:00
```

- Transfer from `remote` to local computer (using `~` is only difference):

```
$ scp mpm288@blargh.ru:~/colorize.sh /usr/share/
colorize.sh                                   100% 3299     3.2KB/s   00:00
```

- Transfer from the `client` to the `remote` (just reverse it):

```
$ scp /usr/share/colorize.sh mpm288@blargh.ru:~/Desktop/
colorize.sh                                   100% 3299     3.2KB/s   00:00
```

- As with regular `cp`, can give a new name at same time:

```
$ scp /usr/share/colorize.sh mpm288@blargh.ru:~/new_name.sh
colorize.sh                                   100% 3299     3.2KB/s   00:00
```

More Git stuffs!

Staging and you

- Go to a git repo, create `file`
- run `git status`

```
$ git status

On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include...

        file

nothing added to commit but untracked files present
```

Tracked and untracked

- files are *tracked* when they have been committed to the repo at some point
- files are *untracked* when they have *never* been committed to the repo
- files are *staged* when the are *about* to be committed to the repo

```
$ git add file
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   file
```

staging files

```
$ git commit -m 'new file'
[master (root-commit) b68fe41] new file
 1 file changed, 1 insertion(+)
 create mode 100644 file
$ echo more text >> file
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes ...

        modified:   file

no changes added to commit (use "git add"/"git commit -a")
```

staging files

- Files you edit are *not automatically* staged
    - `git commit -m` won't include them
- `git commit -a -m` says "stage everything, then commit"
- `git add <file>` says "stage this one file"
    - can `git add` everything, then `git commit -m` when done

Working remotely with git

- you and your partner want to collaborate *with* wash
  - Easy! `clone` your partner's repo, then `pull` updates from each other!
- you and your partner want to collaborate *without* wash
  - Problem: where do you `pull` from?
- solution 1: SSH URLs!
  - can `pull` from `username@machine:path`
  - only works if you can reach the machine
- problem! I have a laptop! It's behind a firewall.
  - there's no stable URL or IP address to pull from...

Bare git repos and the glory of github

- Solution: find *one* machine with a URL
  - put a `bare` repository on there
  - have everyone synchronize via that repository with `git push`

Send repo contents to bare remote

```
git push <url>
```

- A `bare` repository acts as a *mirror*
  - `push` leaves some data there,
  - `pull` finds the data later.
- `git init --bare` to create

An example: working remotely via wash

- initialize a bare repository on wash...

```
$ git init --bare ~/course/cs2043/repo
Initialized empty Git repository in repo/
```

- and clone this repository to your local computer

```
$ git clone milano@wash.cs.cornell.edu:course/cs2043/repo/
Cloning into repo...
warning: You appear to have cloned an empty repository.
done.
$ touch file && git add file && git commit -m 'msg' file
$ git push
```

Terminal Multiplexing

What is Multiplexing?

- Complex combinatorial logic meant to be studied with rigor and painful effort.
  - But not in this class!
- Terminal multiplexing is just the ability to:
  - Split your terminal into multiple *panes*.
  - Be able to *detach* and re-*attach* to a `shell` without having to close it.
  - A whole lot more, but we will focus on these.
- You can leave your multiplexed terminal running on the `remote`, and connect to it from any `client` you want, whenever you want.
- Extremely convenient if you want to be able to work effectively with `ssh`.
- Available on `ugclinux`!

Suggested Multiplexer: `tmux`

Terminal Multiplexer

`tmux [options]`

- `tmux` (with no options) starts a new multiplexed instance.
- Can split into *panes* horizontally and vertically.
- Can `tmux detach` (put in "background", it's still running).
- Can `tmux attach` to bring to "foreground" again.
- Can open new windows, sessions, panes, and more.
    - Use `tmux list-*` commands for active info:
        - `list-buffers`, `list-clients`, `list-commands`, `list-keys`, `list-panes`, `list-sessions`, `list-windows`.
- Use `ctrl+D` to close current *in-focus* pane / window.
    - If you close the last pane of a session, that session ends.

Brief Notes on Multiplexing with `tmux`

- Learn the hotkeys: http://tmuxcheatsheet.com/
- After you `ssh` in, just `tmux attach` to open top-level session.
  - Not sure which session? `tmux ls`, then `tmux attach -t <num>`
- Where is my mouse?!!!
  - Use `shift+click` to highlight with your mouse.
  - May want to bring the current *pane* to full-screen temporarily with `<cmd-seq>+Z`.
    - `<cmd-seq>` is `ctrl+B` by default, but can change it.
    - Un-fullscreen with another `<cmd-seq>+Z`.

Further `tmux` Customization

- Configurations go in a “dotfile”: `~/.tmux.conf`
- Save your layouts with `teamocil`!
  - `gem install teamocil`
  - See http://www.teamocil.com/ for more information.
- First run `tmux`, then launch `teamocil <name>`

References

[1] Stephen McDowell, Bruno Abrahao, Hussam Abu-Libdeh, Nicolas Savva, David Slater, and others over the years. “Previous Cornell CS 2043 Course Slides”.