Contents

1 FIJI Tips and Tricks

There are some easy changes we can make to FIJI to let it be more user friendly. Today we're going to go over a bunch of useful things. We'll start off with some useful configurations.

1.1 Bash

In Bash, whenever you start a shell, a file called `~/.bashrc` (bash run config) is loaded automatically. You can use this to store settings and aliases that you want to have available at all times. Some things you can customize:

- Your command line prompt
- Default environment variables
- Aliases for commands
- Keybindings for your terminal

I've provided a sample containing some things out of my `~/.bashrc` file below. We'll walk through what each line does.

```
# All of this should go in ~/.bashrc

# First, we'll change the default prompt that we use for bash. I've
# provided the one that I use on FIJI below, but you can customize
# your prompt in a few different ways. The variable PS1 is a string
# describing your prompt. If you want to make your own, look at a site
# like http://ezprompt.net/ or
# https://scriptim.github.io/bash-prompt-generator/
```

```
# You can put a lot of information into your command line prompt, and
# I think something like this is a lot more helpful than the default.
export PS1="\n\[\e[32;1m\]\u@\[\e[32;1m\]\h:\[\e[36;1m\]\w \[\e[37;0m\]:: \[\e[33;1m\]\

# Next, you can change the default editor used by bash for its
# internal commands. For example, if you prefer nano or emacs you
# could replace vim here with either of those. This will come in handy
# later when we start going over keybindings.
export VISUAL=/usr/bin/vim
export EDITOR=/usr/bin/vim

# By default, bash uses the readline library, which provides
# emacs-like keybindings. If you prefer, set your terminal to vi mode
# and you'll be able to use treat your command line like you were in
# vim.
set -o vi

# Finally, aliases are basically "shortcuts" that let you have short
# names for more complex commands. These can save you lots of time if
# you're working at the terminal a lot. I've provided a few that I
# find useful below.
alias ls='ls --color=auto ' # Make ls colored by default
alias grep='grep --color=auto ' # Make grep colored by default
alias ll='ls -lah' # This is something that I use all the time
alias us='source ~/.bashrc' # Reload bashrc. "update shell"
alias s='git status' # Git status, since it's called frequently
alias squ='squeue -u $(whoami)' # Your own sbatch jobs
```

1.2 Useful Shell Keybindings

When working in the shell, there are several keybindings and shortcuts that you can use to speed up your work. Bash uses a library called readline, which defaults to emacs keybindings. If you keep it in **emacs mode**, you can use some of the following keybindings:

Keybinding	Function
C-a	Start of line
C-e	End of line
A-f	Next Word
A-b	Previous Word
C-r	Reverse Command Search
C-x C-e	Edit command in EDITOR
C-l	Clear the screen

For more keybindings, check out the bindings for emacs

If you choose to use **vi mode**, you can use most common vim commands at the command line. To exit insert mode, use `ESC`. Then, you can edit using vim commands as you see fit. In this mode, use the key `v` to edit your command in EDITOR.

One of the most useful things you can do here is edit a the command in your terminal using your editor. This gives you all the features of a text editor and lets you do things like write multi-line commands without having to go straight to a script. To do this, just use `C-x C-e` (emacs mode) or `ESC v` (vi mode). Your current command will be opened in your editor. When you're finished with your command, just save and quit and the command will be executed in your terminal.

The other essential keybinding to know is `C-r`, which lets you look up previous commands from your history. Say you used a long command earlier and did a bunch of other work after that. If you wanted to use the command again, you could use the arrow keys to look through your history until you find it, but that could take a while. If you use `C-r` instead, you can just search for it in your history instead.

1.3 Screen

When you log in to FIJI it can be useful to keep a persistent terminal session going so that you can pick up where you left off. One option for this is a utility called `screen`, which allows you to keep one or more terminal sessions on the server at a time.

IMPORTANT: To access the same session every time, you need to log in to the same login node. Pick either fiji-1 or fiji-2, and log in to fiji-1.colorado.edu or fiji-2.colorado.edu instead of just fiji.colorado.edu.

First, let's create a configuration file for screen. We'll put all this configuration into `~/.screenrc`, and then learn how to work with screen.

```
# All of this should go in ~/.screenrc
```

```
# First, we configure the status line for screen. By default, one
# isn't provided, so we have to define it ourselves. Like with bash,
# we configure how we want it to look using a string (in this case
# called hardstatus). These 2 lines will give us a status bar at the
# bottom showing the status of our screen session.
backtick 0 30 30 sh -c 'screen -ls | grep --color=no -o "$PPID[^[:space:]]*"'
hardstatus alwayslastline
hardstatus string '%{= kW}[%{y}%H:%`%{W}][%= %{= kw}%-w%{+b ck} %n*%t%?(%u)%? %{-}%+w %

# This command tells screen to load bash in a user shell by default,
# which means that we'll get all the configuration we did in our
# bashrc across all screen sessions.
defshell -bash

# This command tells screen to stop saving history when we're using a
# tool that takes up the whole screen like vim or less. This means
# that we won't ruin our ability to scroll back using screen's
# built-in history.
altscreen on

# This command tells screen to automatically detach if the terminal
# connection on your local machine dies or is killed. This means that
# you'll keep your session even if you accidentally kill a terminal or
# lose your internet connection.
autodetach on

# This command defines how much history we want to keep for each
# terminal screen manages. Here I've chosen to keep 5000 lines of
# history, which should be more than enough for almost any purpose.
defscrollback 5000
```

Now that we have everything configured, let's learn about how screen actually works. When you first log into the head node, you can start a screen session using the command `screen`. When you do this, you should see the status bar that we just configured pop up as the last line in your terminal. When you're using screen, you can run screen-specific commands using keybindings under `CTRL-a`. I've made a small table of the most useful ones below.

Keybinding	Name	Description
C-a c	create	Make a new terminal
C-a d	detach	Detach (get out) of screen
C-a 4	switch	Switch to terminal 4 (can use any number)
C-a A	rename	Rename your current terminal
C-a [	scroll back	Enter screen's history to scroll backwards

These are the commands I think are most useful for working with screen. If you want to know more of them, check out the keybinding documentation.

Another useful feature is the ability to split screen sessions vertically and horizontally. Below is a brief table of the keybindings used for splitting vertically and horizontally:

Keybinding	Name	Description
C-a |	vertical split	Split current window vertically
C-a S	horizontal split	Split current window horizontally
C-a TAB	switch split	Change to the next split window
C-a "	list windows	List all available windows you have open
C-a X	close window	Close the current window
C-a Q	destroy splits	Reset any splitting you've done to 1 window

The other important thing to know about screen is that it handles your terminals using "sessions", which you can have one or more of. To see sessions you have active, type in `screen -ls`. To attach to an existing session or make a new one, use `screen -DR $SESSION_NAME`. If the session exists you'll be reconnected to it. If not, a new session with that name will be created.

1.4 Less Common Commands

1.4.1 Parallel

GNU Parallel is a command for running tasks in parallel that comes installed by default on FIJI. The advantage of using something like parallel is that it only creates as many jobs as there are processors available (respecting your slurm settings). For example:

```
# Unzip every .gz file in a directory in parallel. {} is where the
# name of each file is substituted.
parallel gunzip {} :: *.gz
```

Parallel has too many features for me to cover in detail here. I highly recommend it for any time you need to run multiple commands at the same time.

A good introductory example can be found here, and a much more detailed introductory guide can be found here.

1.5 Special Extras (advanced)

If you want to work with more advanced shell functions, there are a few resources and tools I recommend:

- The Art of the Command Line
- Pure Bash Bible
- FZF - An Advanced Fuzzy Searching Tool
- TMUX - A More Advanced Version of Screen (for your local machine)
- Google's Shell Style Guide (for writing scripts)