Exercises

Job control

From what we have seen, we can use some `ps aux | grep` commands to get our jobs' pids and then kill them, but there are better ways to do it. Start a `sleep 10000` job in a terminal, background it with `Ctrl-Z` and continue its execution with `bg`. Now use `pgrep` to find its pid and `pkill` to kill it without ever typing the pid itself. (Hint: use the `-af` flags).

```
# Start a process
~$ sleep 10000
# Pause the job in the background
^Z
[1]+  Stopped                 sleep 10000
# Show status of jobs
~$ jobs
[1]+  Stopped                 sleep 10000
# Resuming the process in the background
~$ bg %1
[1]+ sleep 10000 &
~$ jobs
[1]+  Running                 sleep 10000 &
~$ man pkill
# Find the PID of the backgrounded job
~$ pgrep -af "sleep 10000"
12934 sleep 10000
# Kill the process based on its full command
~$ pkill -f "sleep 10000"
[1]+  Terminated              sleep 10000
```

Say you don't want to start a process until another completes, how would you go about it? In this exercise our limiting process will always be `sleep 60 &`. One way to achieve this is to use the `wait` command. Try launching the sleep command and having an `ls` wait until the background process finishes.

```
# Sleep for 60s in the background
~$ sleep 60 &
# Wait until the process finishes before executing ls
~$ wait $! && ls
```

However, this strategy will fail if we start in a different bash session, since `wait` only works for child processes. One feature we did not discuss in the notes is that the `kill` command's exit status will be zero on success and nonzero otherwise. `kill -0` does not send a signal but will give a nonzero exit status if the process does not exist. Write a bash function called `pidwait` that takes a pid and waits until the given process completes. You should use `sleep` to avoid wasting CPU unnecessarily.

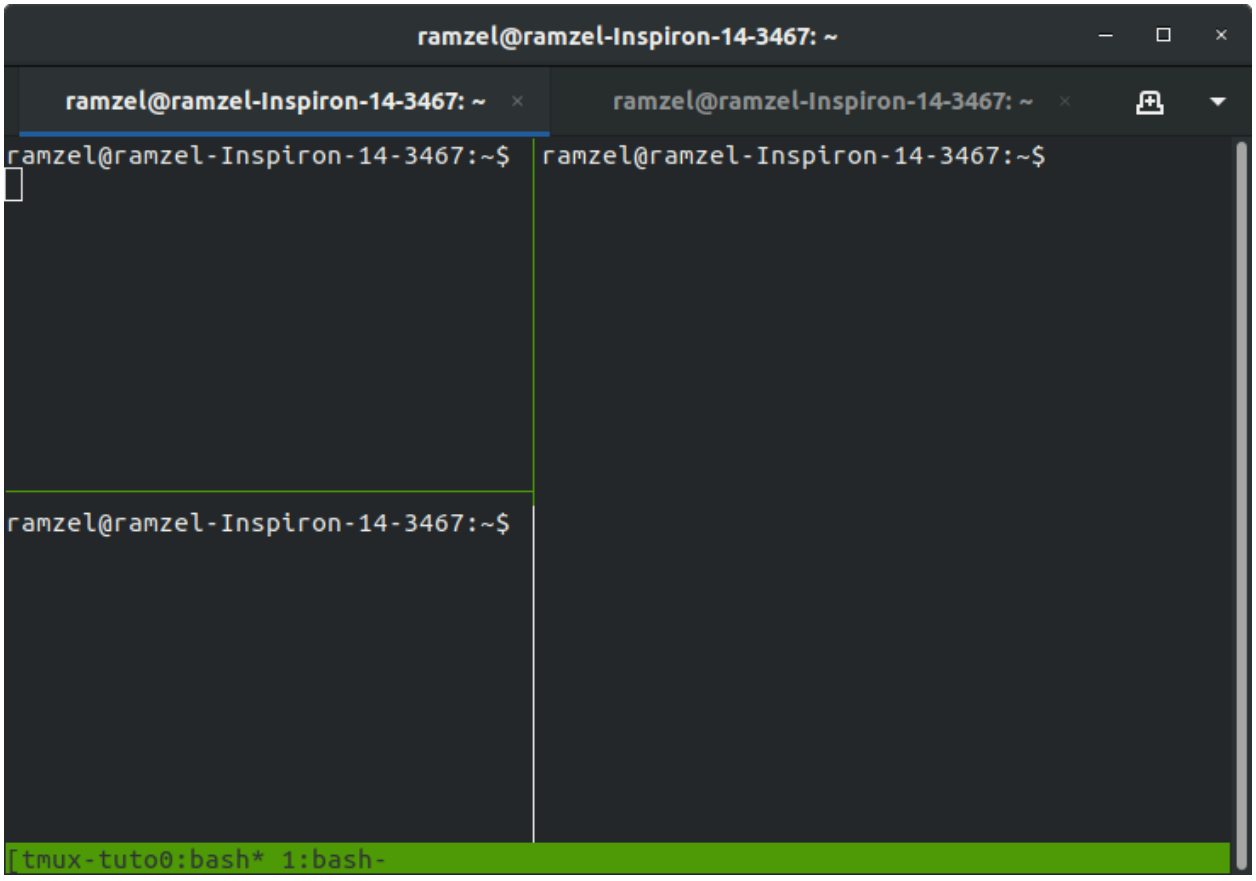
```
~$ vim pidwait.sh
#!/usr/bin/env bash

pidwait () {
      # Wait until the process finishes
      while kill -0 $1 2>/dev/null; do
            sleep 1
      done
      # List current directory contents
      ls
}
~$ source pidwait.sh
~$ sleep 60 &
# Execute ls after 60s
~$ pidwait "$(pgrep -f 'sleep 60')"
```

Terminal multiplexer

Follow this `tmux` tutorial and then learn how to do some basic customizations following these steps.

```
~$ tmux
[detached (from session 0)]
~$ tmux ls
0: 2 windows (created Tue May  4 14:40:18 2021) [80x23]
~$ tmux attach -t 0
~$ tmux rename-session -t 0 tmux-tutorial
~$ tmux ls
tmux-tutorial: 2 windows (created Tue May  4 14:40:18 2021) [80x23]
~$ tmux attach -t tmux-tutorial
```



Aliases

Create an alias `dc` that resolves to `cd` for when you type it wrongly.

```
~/Documents$ alias dc=cd
~/Documents$ dc ..
~$
```

Run `history | awk '{$1="";print substr($0,2)}' | sort | uniq -c | sort -n | tail -n 10` to get your top 10 most used commands and consider writing shorter aliases for them.

```
~$ history | awk '{$1="";print substr($0,2)}' | sort | uniq -c | sort
-n | tail -n 20
    6 git status
    12 cd ..
    23 cd -
    81 ls
    ...
~$ vim ~/.bash_aliases
...
# Some aliases
alias gs='git status'
alias up='cd ..'
alias back='cd -'
alias sl=ls
```

Dotfiles

Create a folder for your dotfiles and set up version control.

```
~$ mkdir ~/configs
~$ cd ~/configs/
~/configs$ git init
Initialized empty Git repository in /home/ramzel/configs/.git/
```

Add a configuration for at least one program, e.g. your shell, with some customization (to start off, it can be something as simple as customizing your shell prompt by setting `$PS1`).

```
~/configs$ vim bashrc
# Shorten Bash prompt to just the last directory
if [ "$color_prompt" = yes ]; then

PS1='${debian_chroot:+($debian_chroot)}\[\033[01;32m\]\u@\h\[\033[00m
\]:\[\033[01;34m\]\W\[\033[00m\]\$ '
else
    PS1='${debian_chroot:+($debian_chroot)}\u@\h:\W\$ '
fi

# Aliases
if [ -f ~/.bash_aliases ]; then
    source ~/.bash_aliases
fi

# History management
export HISTCONTROL=ignoreboth
export HISTSIZE=5000
export HISTIGNORE="clear:bg:fg:cd:cd -:cd ..:exit:date:w:*
--help:ls:l:ll:lll"
...
```

I also added configurations for vim and tmux.

Set up a method to install your dotfiles quickly (and without manual effort) on a new machine. This can be as simple as a shell script that calls `ln -s` for each file, or you could use a specialized utility.

Using Dotbot:

```
~/configs$ git submodule add https://github.com/anishathalye/dotbot
Cloning into '/home/ramzel/configs/dotbot'...
~/configs$ git config -f .gitmodules submodule.dotbot.ignore dirty
~/configs$ cp dotbot/tools/git-submodule/install .
```

```
~/configs$ vim install.conf.yaml
[
    {
        "link": {
            "~/configs": ""
        }
    }
]
...
```