Command-line Environment

Job Control

- Interrupt jobs with `Ctrl-C`
- `Ctrl-C` sends `SIGINT` signal
- `Ctrl-\` sends `SIGQUIT` signal

Killing a process

- Signals are *software interrupts*
- Example: Python program that ignores `SIGINT`

```python
import signal, time

def handler(signum, frame):
    print("\nI got a SIGINT, but I am not stopping")

signal.signal(signal.SIGINT, handler)
i = 0
while True:
    time.sleep(.1)
    print("\r{}".format(i), end="")
    i += 1
```

- Use `kill -TERM <PID>` for `SIGTERM`

Pausing and backgrounding processes

- `Ctrl-Z` sends `SIGTSTP`
- Use `fg` or `bg` to continue jobs
- `&` runs command in background
- `nohup` or `disown` to prevent closing terminal from killing process

```
$ sleep 1000
^Z
$ bg %1
$ jobs
$ kill -STOP %1
$ kill -SIGHUP %1
$ kill %2
```

- `SIGKILL` is uncatchable and terminates process immediately

Terminal Multiplexers

- Run multiple shell sessions with tools like `tmux`
- `tmux` keybindings start with `<C-b>`

`tmux` Hierarchy

- Sessions
  - Create, list, detach, attach
- Windows
  - Create, navigate, rename, list
- Panes
  - Split, navigate, zoom, copy mode

Aliases

- Short form for long commands
- Example: `alias ll="ls -lh"`
- Aliases need to be in shell startup files to persist

```
alias gs="git status"
alias v="vim"
alias sl=ls
alias mv="mv -i"
alias df="df -h"
alias la="ls -A"
alias lla="la -l"
```

Dotfiles

- Configuration files for tools
- Common dotfiles: `.bashrc`, `.vimrc`, `.ssh/config`, `.tmux.conf`
- Organize in a version-controlled folder
- Symlink into place

Portability

- Use if-statements for machine-specific settings
- Use includes for shared configurations

Remote Machines

- Use `ssh` for secure remote access
- Execute commands directly with `ssh`
- Use `ssh-keygen` for key-based authentication

```
ssh foo@bar.mit.edu
ssh foobar@server ls
cat .ssh/id_ed25519.pub | ssh foobar@remote 'cat >> ~/.ssh/authorized_keys'
```

Copying files over SSH

- Use `ssh+tee`, `scp`, or `rsync`
- `scp` syntax: `scp local_file remote_host:remote_file`
- `rsync` for efficient copying

Port Forwarding

- Local and Remote Port Forwarding
- Example: `ssh -L 9999:localhost:8888 foobar@remote_server`

SSH Configuration

- Use `~/.ssh/config` for aliases and settings
- Server side config in `/etc/ssh/sshd_config`

Shells & Frameworks

- `zsh` is a superset of `bash`
- Frameworks like `prezto` or `oh-my-zsh` enhance shell experience
- `fish` includes user-friendly features by default

Terminal Emulators

- Choose based on personal preference and workflow needs
- Consider font, color scheme, shortcuts, and performance

Terminal Emulators (Cont.)

- Customize your terminal emulator settings for:
  - **Font choice**: Choose a readable and pleasant font.
  - **Color Scheme**: Select colors that are easy on the eyes.
  - **Keyboard shortcuts**: Optimize for efficiency.
  - **Tab/Pane support**: Enable easy navigation.
  - **Scrollback configuration**: Set your history preferences.
  - **Performance**: Consider GPU-accelerated terminals like Alacritty or kitty.

Invest time in setting up your terminal, it's your main workspace!

Conclusion

- Improving your command-line environment can significantly enhance productivity.
- Understand job control and signals for effective process management.
- Terminal multiplexers like `tmux` offer powerful session management.
- Aliases and dotfiles save time and ensure a consistent environment.
- Remote machine management via SSH is essential for modern computing tasks.
- Explore different shells and frameworks to find what works best for you.
- Customize your terminal emulator to make your work enjoyable and efficient.

Happy coding, and may your command line be ever in your favor!