

Tools, Workflow and Resources

About Lab3 HSRW

Lab3



IoTlab

internet
of things
lab

More info: [IoT Lab](#)

Dronelab

drones and
robots
lab

More info: [IoT Lab](#)



EOlab

earth
observation
lab

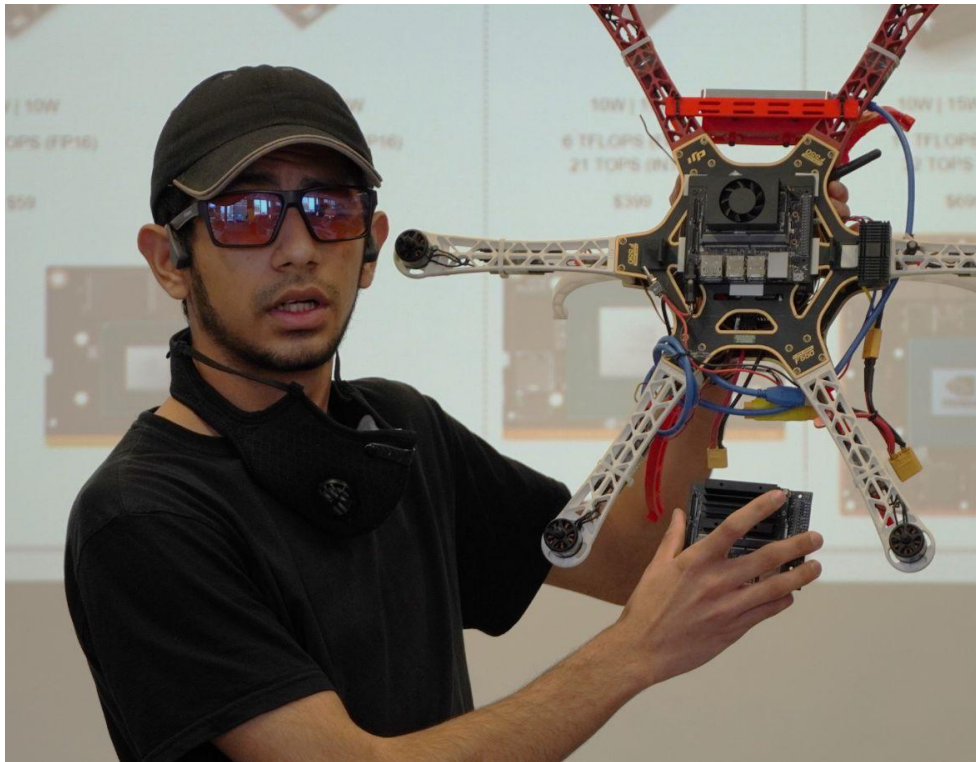
More info: [EO Lab](#)

About Me

Harley Lara

Mechatronics Engineer

- Robotics and Drones Integration
- Software and hardware dev
- Machines design



About Workshop

Basics Tools

- Terminology
- Shells
- Terminal emulators
- Terminal multiplexers
- Scripting for integration

Manage configs

- Dotfiles
- SSH Keys
- Automation
- Tiling window manager (bonus)

Documentation (of your work)

- Markups
- Hosting Docs
- Diagrams
- Screenshots

Disclaimer: In this workshop we will not cover anything related to code editors or code editor extensions.

About Workshop

Basics Tools

- Terminology
- Shells
- Terminal emulators
- Terminal multiplexers
- Scripting for integration

Manage configs

- Dotfiles
- SSH Keys
- Automation
- Tiling window manager (bonus)

Documentation (of your work)

- Markups
- Hosting Docs
- Diagrams
- Screenshots

Extras resources for coding

- Design Patterns
- Refactoring
- Git branching strategies
- Style Guides

Let's Start

>> **launch workshop**

Basics1

First part of the workshop

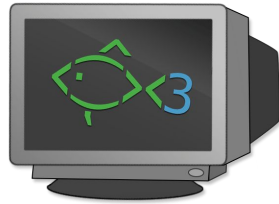
Terminology

Shell != Terminal != Console

Shells



[Bash](#)



[Fish](#)

KORNSHELL

[Ksh](#)



[Zsh or Z Shell](#)

Terminal Emulators

- [Konsole](#): KDE's desktop environment
- [kitty](#) (not [KiTTY](#)): GPU based terminal emulator
- [Alacritty](#) (cross-platform): GPU based terminal emulator
- [Terminator](#): GNOME's desktop environment
- [PuTTY](#) (for windows): Mainly ssh and telnet connection
- [Windows Terminal](#) (for window)
- Find more [here](#)

To the Terminal and Beyond



Terminal Multiplexers

Terminal Multiplexers

Why don't just
use Tabs?

Because it depends on a graphical environment

```
>> launch demo
```

Terminal Multiplexers

Create a new session

```
> screen -S mysession
```

Detach session

```
> `C-a d`
```

List of sessions

```
> screen -ls
```

Attach session

```
> screen -r mysession
```

Split display horizontally

```
> `C-a S`
```

Split display vertically

```
> `C-a |`
```

Jump to next display

```
> `C-a tab`
```

Remove current region

```
> `C-a X`
```

Create a window

```
> `C-a c`
```

GNU screen

>> **launch demo**

Terminal Multiplexers

The main
problem of
screen

It does not provide graphical feedback

Tmux



Terminal Multiplexers

A little bit of
terminology
first !!

Terminal Multiplexers

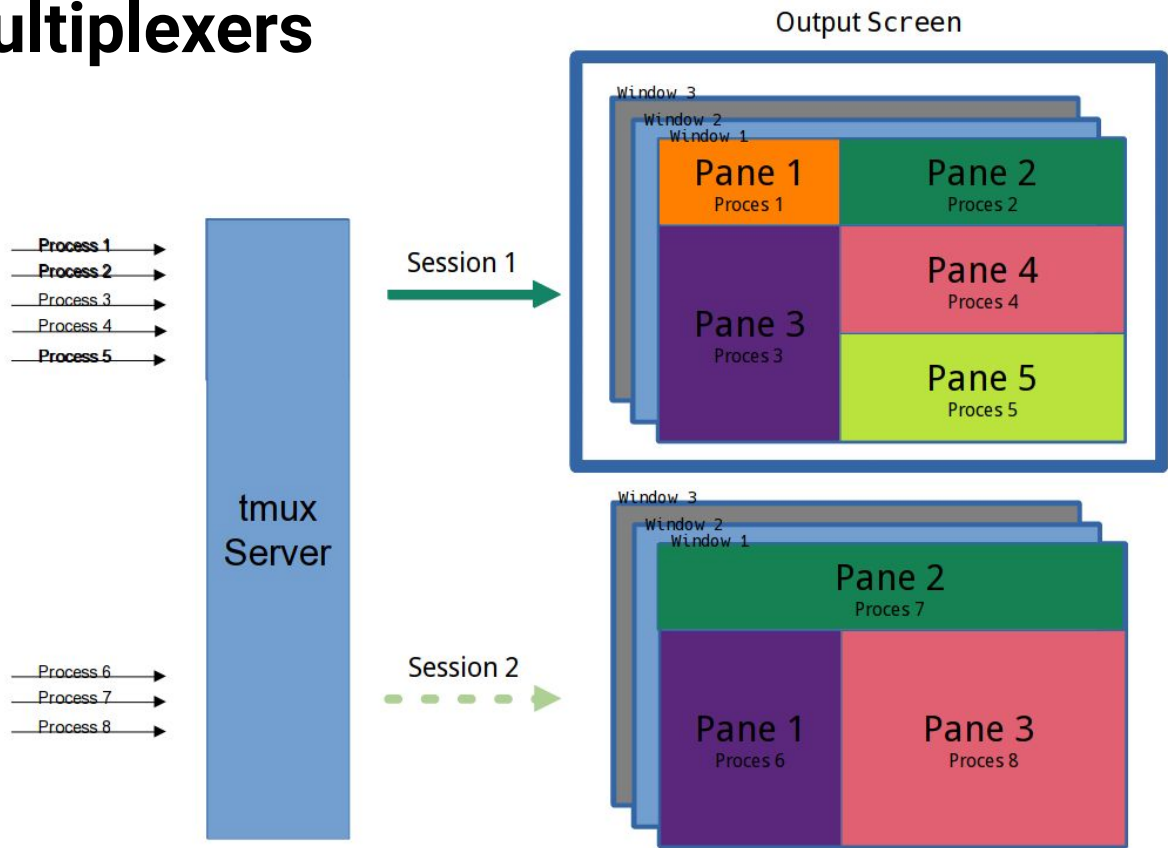


Image source: arcolinux.com

Terminal Multiplexers

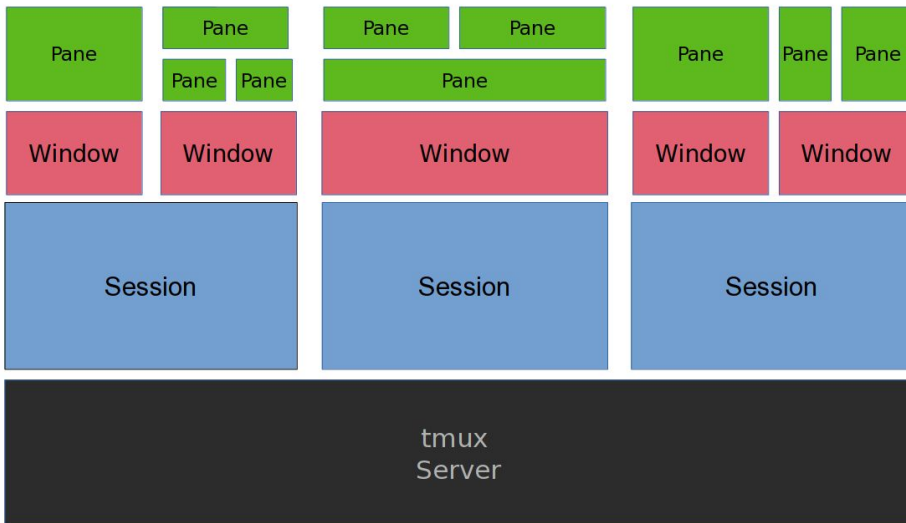


Image source: [archlinux.com](https://www.archlinux.com)

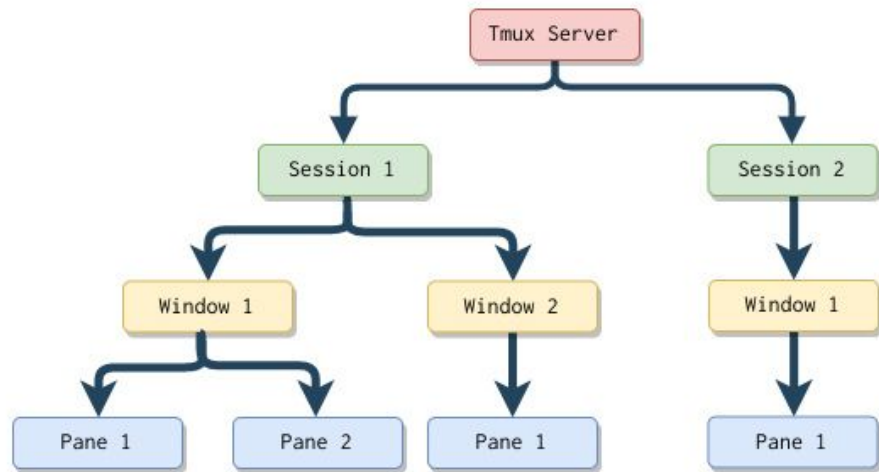


Image source: thevaluable.dev

Terminal Multiplexers

Demo Time

```
>> tmux
```

Terminal Multiplexers

Run new session

> tmux

Run new session with name

> tmux new -s mysession

Detach

`C-b d`

List all sessions

> tmux ls

Attach session

> tmux a -t mysession

Next session

`C-b)`

Previous session

`C-b (`

Create window

`C-b c`

Move to next window

`C-b n`

Move to previous window

`C-b p`

Panes

Vertical Split

`C-b %`

Horizontal split

`C-b ``

Move to pane

`C-b arrow`

Scripting for integration

Let's make
something
cool

Prerequisites:

- [Tmux](#)
- [fzf](#) (A command-line fuzzy finder)

Scripting for integration

A simple example first

```
>> hello
```

A hello bash script

Scripting for integration

```
#!/bin/bash

DIRECTORIES=$(cat ~/.tmux-dirs | tr "\n" " ")

if [[ $# -eq 1 ]]; then
    SELECTED=$1 && [[ "$SELECTED" == '.' ]] && SELECTED="$PWD"
else
    SELECTED=$(find $DIRECTORIES -mindepth 1 -maxdepth 1 -type d | fzf)
fi

if [[ -z $SELECTED ]]; then
    exit 0
fi

SELECTED_NAME=$(basename "$SELECTED" | tr . _)
SELECTED_NAME=${SELECTED_NAME:0:8}

if [[ -n $TMUX ]]; then
    # inside tmux
    tmux switch-client -t "$SELECTED_NAME" || \
    tmux new-session -ds "$SELECTED_NAME" -c "$SELECTED" && \
    attach -t "$SELECTED_NAME"
fi
```

Scripting for integration



```
#!/bin/bash

# Bash profile
[[ -f ~/.bashrc ]] && . ~/.bashrc

addToPath() {
    if [[ "$PATH" != *"$1"* ]]; then
        export PATH=$PATH:$1
    fi
}

addToPath $HOME/.local/bin

bind '"\C-f": "\C-k \C-utmux-sessionizer\n"'
bind '"\C-h": "\C-k \C-ucht.sh\n"'
```

Manage configs²

Second part of the workshop

Dotfiles

Dotfiles

```
>> source ~/.bashrc
```

User-specific application configuration is traditionally stored in so called dotfiles (files whose filename starts with a dot).

Dotfiles

Exploring some dotfiles

```
>> cat ~/.bashrc
```

Ssh keys

SSH keys

Secure Shell Protocol

Example: [link](#)

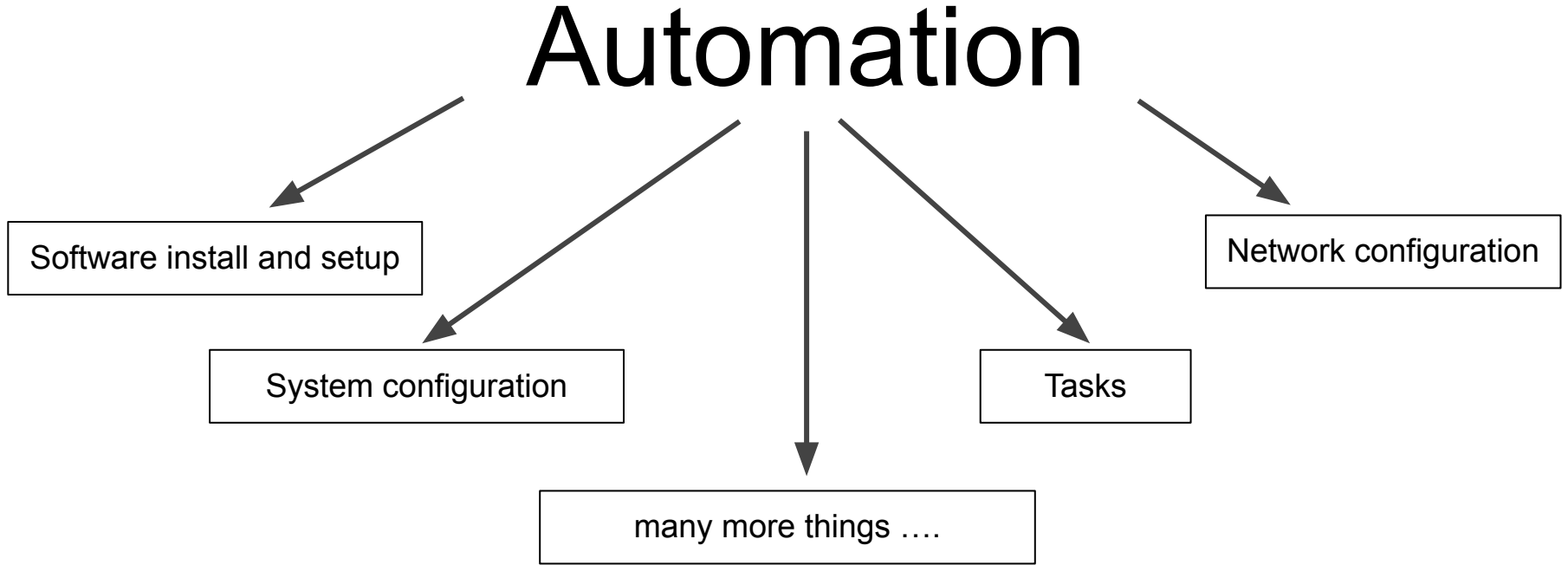
Bad practice:

- Plain text in your computer
- Plain text in [Github](#) !!! (internet)
- Plain text in emails

- Generate new SSH keys per system and service
- Store the keys in an encrypted format
- Use tools for SSH keys management

Never use plain text to storage ssh keys!!

Automation



Automation

Bash scripting for automation?

```
>> bash my_script.sh
```


Automation

Ansible

```
>> ansible-playbook programs.yml
```

Ansible is a suite of software tools that enables infrastructure as code.

Automation



```
- host: localhost
  tasks:
    - name: Install my Core System Package
      become: true
      apt:
        name: ["build-essential", "cmake", "pkg-config", "libpthread-stubs0-dev"]
      tags:
        - core
        - software

    - name: Install my tools
      become: true
      apt:
        name: [ "curl", "htop", "python3-pip" ]
      tags:
        - software
        - tools
```

Ansible playbook

Automation



```
#!/bin/bash
```

```
sudo apt-get update && \  
apt-get install -y \  
    build-essential \  
    cmake \  
    pkg-config \  
    libpthread-stubs0-dev
```

```
sudo apt-get install -y \  
    curl \  
    htop \  
    python3-pip
```

Bash script

Automation

```
● ● ●  
  
- host: localhost  
  tasks:  
    - name: Install my Core System Package  
      become: true  
      apt:  
        name: ["build-essential", "cmake", "pkg-config", "libpthread-stubs0-dev"]  
      tags:  
        - core  
        - software  
  
    - name: Install my tools  
      become: true  
      apt:  
        name: [ "curl", "htop", "python3-pip" ]  
      tags:  
        - software  
        - tools
```

Ansible playbook

```
● ● ●  
  
#!/bin/bash  
  
sudo apt-get update && \  
apt-get install -y \  
    build-essential \  
    cmake \  
    pkg-config \  
    libpthread-stubs0-dev  
  
sudo apt-get install -y \  
    curl \  
    htop \  
    python3-pip
```

Bash script

Automation

```
- host: localhost
  tasks:
    - name: Install my Core System Package
      become: true
      apt:
        name: ["build-essential", "cmake", "pkg-config", "libpthread-stubs0-dev"]
        tags:
          - core
          - software
    - name: Install my tools
      become: true
      apt:
        name: [ "curl", "htop", "python3-pip" ]
        tags:
          - software
          - tools
```

Ansible playbook

```
#!/bin/bash

ARG_TAG="-tag"
ARG_HELP="-h"
COLOR_OFF="\033[0m"
COLOR_RED="\033[0;31m"

sudo apt-get update

help(){
  cat << EOF
NAME
  Shell script to install my software

SYNOPSIS
  bash automation2.sh [OPTIONS <string>]

OPTIONS
  -tag    Install software base on tags
         - core: install core Packages
         - tools: install tools
         - software: install all
  -h      Show help
EOF
}

install_core(){
  echo "Install my Core System Packages"
  apt-get install -y \
  build-essential \
  cmake \
  pkg-config \
  libpthread-stubs0-dev
}

install_tools(){
  echo "Install my tools"
  apt-get install -y \
  curl \
  htop \
  python3-pip
}

install_with_tag(){
  if [[ $1 = $ARG_TAG && $2 = "core" ]]; then
    install_core
  elif [[ $1 = $ARG_TAG ]] && [[ $2 = "tools" ]]; then
    install_tools
  elif [[ $1 = $ARG_TAG ]] && [[ $2 = "software" ]]; then
    install_core
    install_tools
  elif [[ $1 = $ARG_HELP ]]; then
    echo "HELP"
    help
  else
    echo -e "${COLOR_RED}ERROR: Invalid tag${COLOR_OFF}"
    help
  fi
}

if [ $# -eq 0 ]
then
  install_core
  install_tools
else
  if [ $# -eq 2 ]; then
    install_with_tag $1 $2
  else
    echo -e "${COLOR_RED}ERROR: Invalid syntax${COLOR_OFF}"
    help
  fi
fi
```

Automation

Ansible for
ssh key :D

Tiling Window Managers (bonus)

Tiling?
window?
managers?

Tiling Window Managers (bonus)

Some Tiling Window Managers

i3

For linux

bug.n

For Windows

Amethyst

For MacOS

Docs3

Third part of the workshop

Markups

Lightweight

Markup

languages

Lightweight Markup Languages are designed to be easy to write using any generic text editor and easy to read in its raw form

- [Markdown](#)
- [MediaWiki](#)
- [DokuWiki](#)
- [reStructuredText](#)
- [AsciiDoc](#)
- [Many others...](#)

Markups



```
# This is a title
```

```
**Lorem ipsum** dolor sit amet, consectetur  
adipiscing elit, *sed do eiusmod* tempor  
incididunt ut labore et dolore magna aliqua.  
Ut enim ad minim veniam, **quis nostrud**  
exercitation ullamco laboris nisi ut aliquip  
ex ea commodo `consequat`
```

- Item
- Other item
- Last item

Markdown



```
<h1>This is a title</h1>
```

```
<p><b>Lorem ipsum</b> dolor sit amet,  
consectetur adipiscing elit, <i>sed do  
eiusmod</i> tempor incididunt ut labore et  
dolore magna aliqua. Ut enim ad minim  
veniam, <b>quis nostrud</b> exercitation  
ullamco laboris nisi ut aliquip ex ea  
commodo <code>consequat</code></p>
```

```
<ul>  
  <li>Item</li>  
  <li>Other item</li>  
  <li>Last item</li>  
</ul>
```

HTML

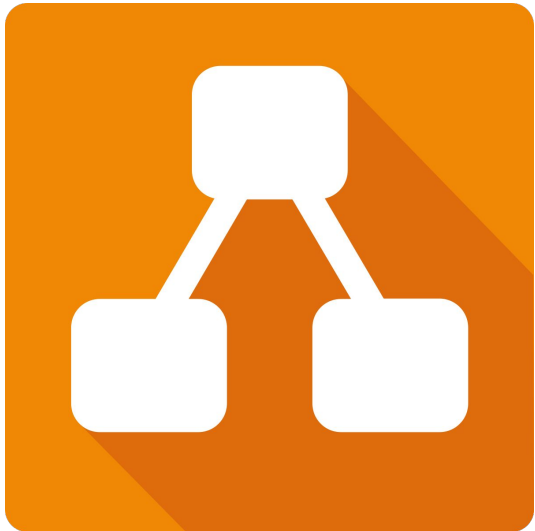
Hosting Docs

Hosting docs

- A readme or set of .MD files in the repository
- Wikis: Doku Wiki, MediaWiki ...
- SSGs (Static Site Generator): GitHub pages, Gitlab pages, etc...

Diagrams

Diagrams



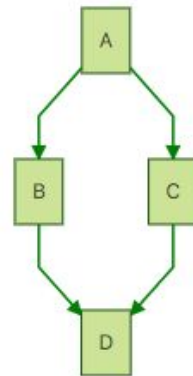
[Draw.io](https://draw.io)

There are not plans to add LaTeX support

Mermaid



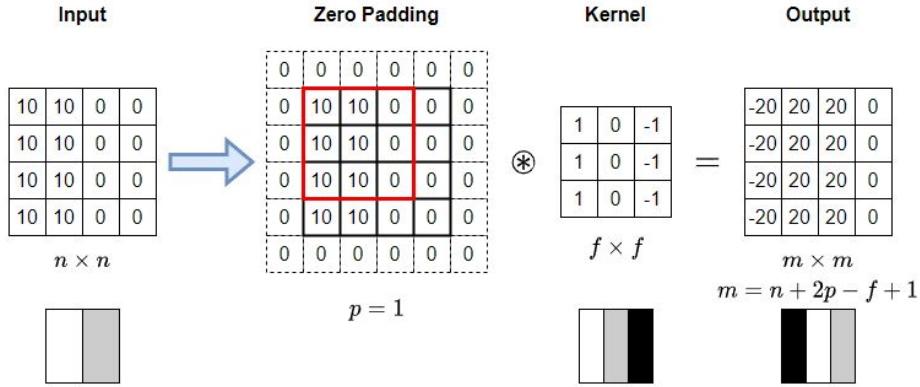
```
graph TD;
  A-->B;
  A-->C;
  B-->D;
  C-->D;
```



Diagrams

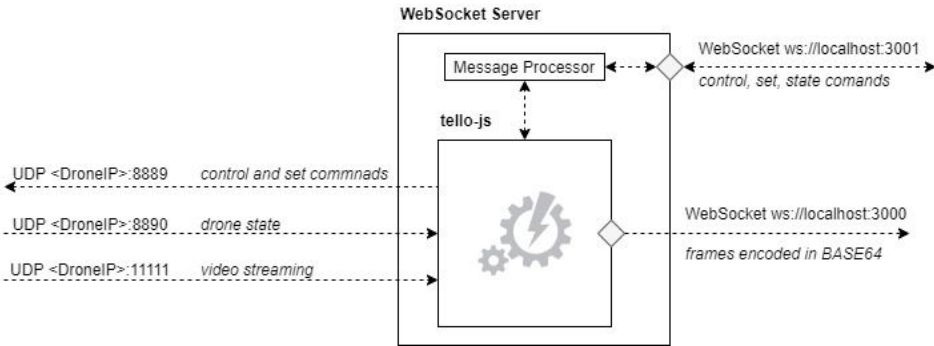
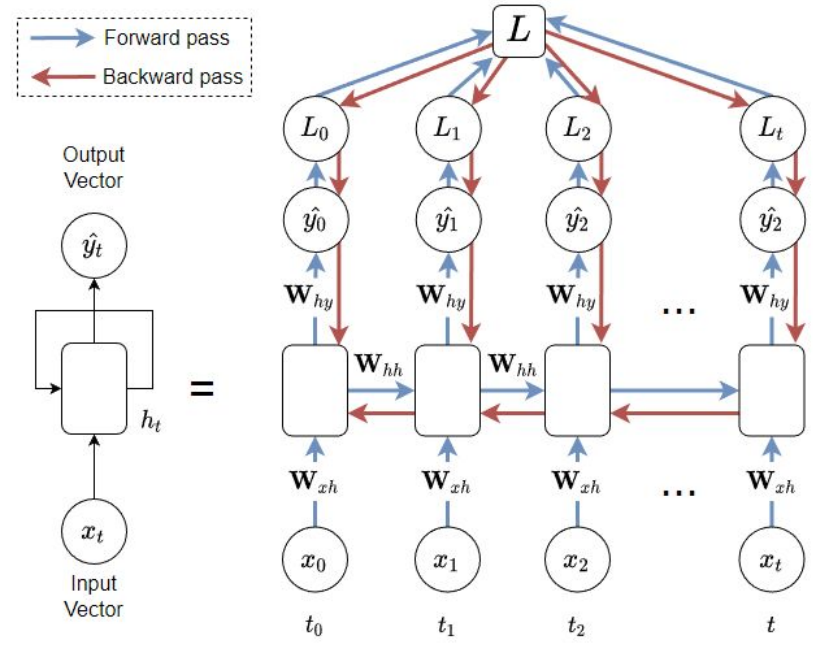
Zero Padding

Same Convolution



Mirror Padding

Backpropagation Through Time (BPTT)



Screenshots ?

- Basics: Text, annotations, shapes
- Add arrows ?
- Number counter ?
- Blur ?

Screenshots



[Flameshot](#)

Resources

Generals

Design patterns:

- [Design patterns by refactoring guru](#)
- [Design patterns infographic](#)

Refactoring

- [Refactoring by refactoring guru](#)

Git branching strategies

- [A successful Git branching model](#)
(GitFlow)
- [GitHub Flow](#)
- [Trunk-based](#)
- [Comparison git flows vs trunk-based](#)

Python development

Coding Style guides

- [PEP 8 – Style Guide for Python Code](#)
- [Google Python Style Guide](#)
- Extra: [PEP 1 – PEP Purpose and Guidelines](#)

Javascript development

Coding Style guides

- [Google JavaScript Style Guide](#)
- [Airbnb JavaScript Style Guide](#)
- [JavaScript Standard Style](#)

C/C++ (for embedded systems)

- [Embedded C Coding Standard](#)

Native Windows commands