Session 4: `tmux`

P. S. Langeslag

8 November 2018

Vim on Your Own System

OS X

Included; open a terminal and enter `vim`

Linux

Included or in package repositories; install the `gvim` package for full clipboard functionality

Windows

Download from http://www.vim.org

Terminal or Graphical?

Whatever works for you

Display Protocols

1984–	X Window System	Still the default display protocol for UNIX-like systems
2008–	Wayland	Abandons 1980s standards in pursuit of greater efficiency and security

Graphical Environments

Desktop Environment (DE)

Complete graphical environment, with windows, icons, sound effects, themes, and graphical interfaces to various tools. (Gnome, KDE, Cinnamon, MATE, Xfce)

Window Manager (WM)

Does what it says and little else. All other DE components can be separately installed.

Display Manager

Graphical application for logging a user into Linux and launching the desktop environment or window manager.

Display Manager: Gnome Display Manager (GDM)

Figure 1: GDM on Ubuntu 10.04 (public domain / WMC)

Desktop Environments: Gnome

Figure 2: Gnome 3 (https://www.gnome.org)

Desktop Environments: KDE

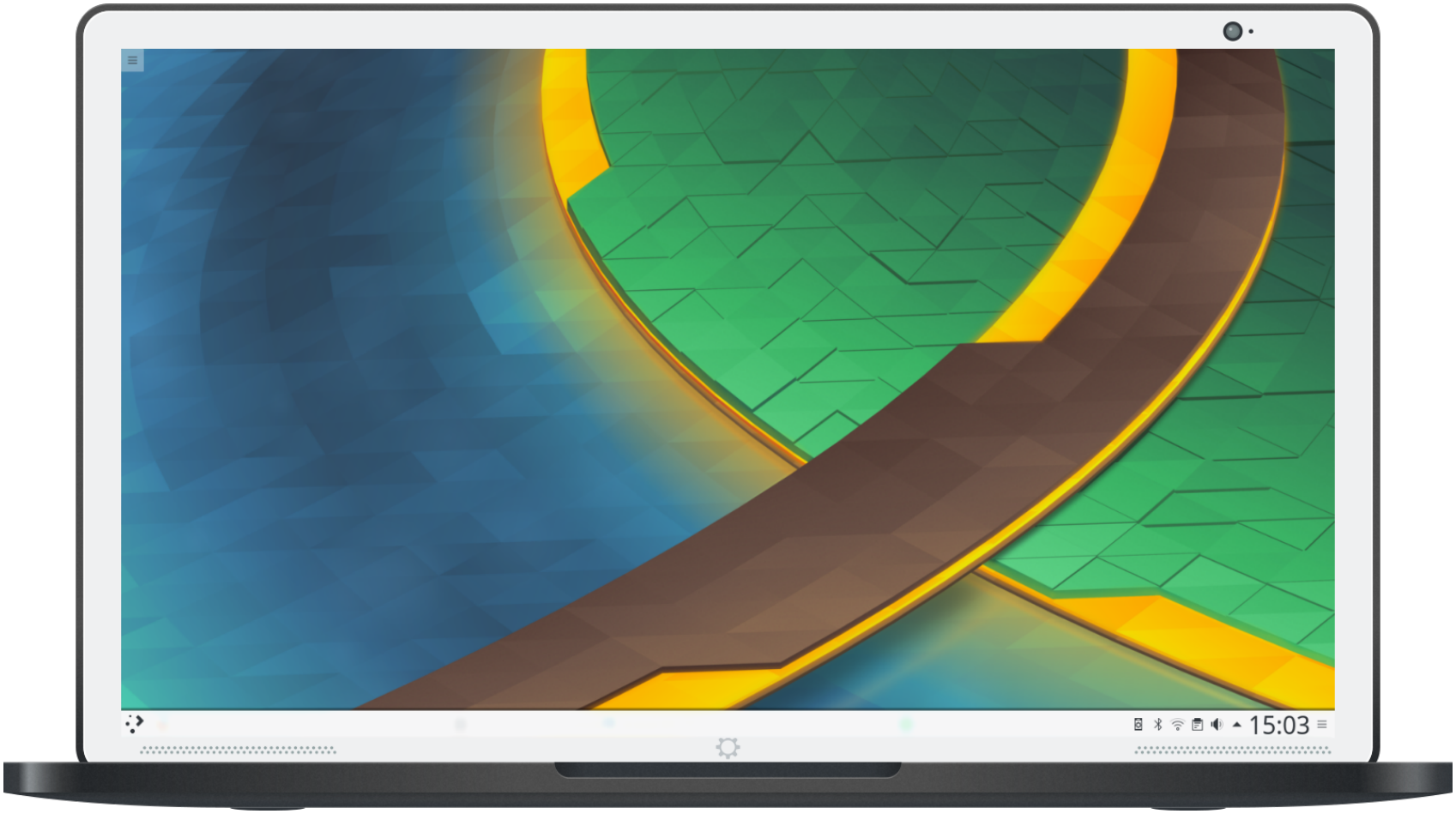

Figure 3: KDE Plasma (https://www.kde.org)

Desktop Environments: MATE

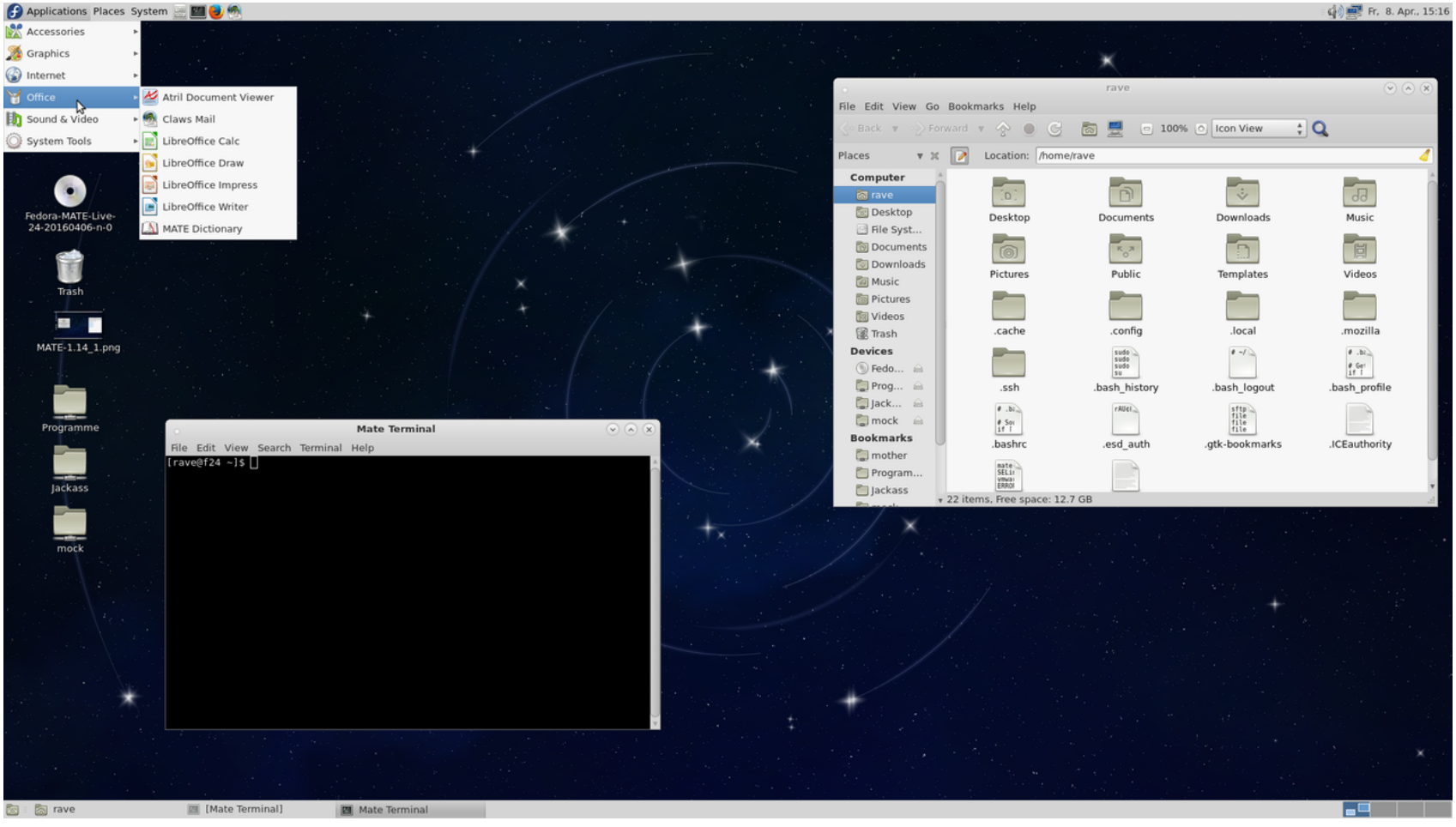

Figure 4: MATE 1.14 (https://www.mate-desktop.org/gallery/)

Desktop Environments: Xfce

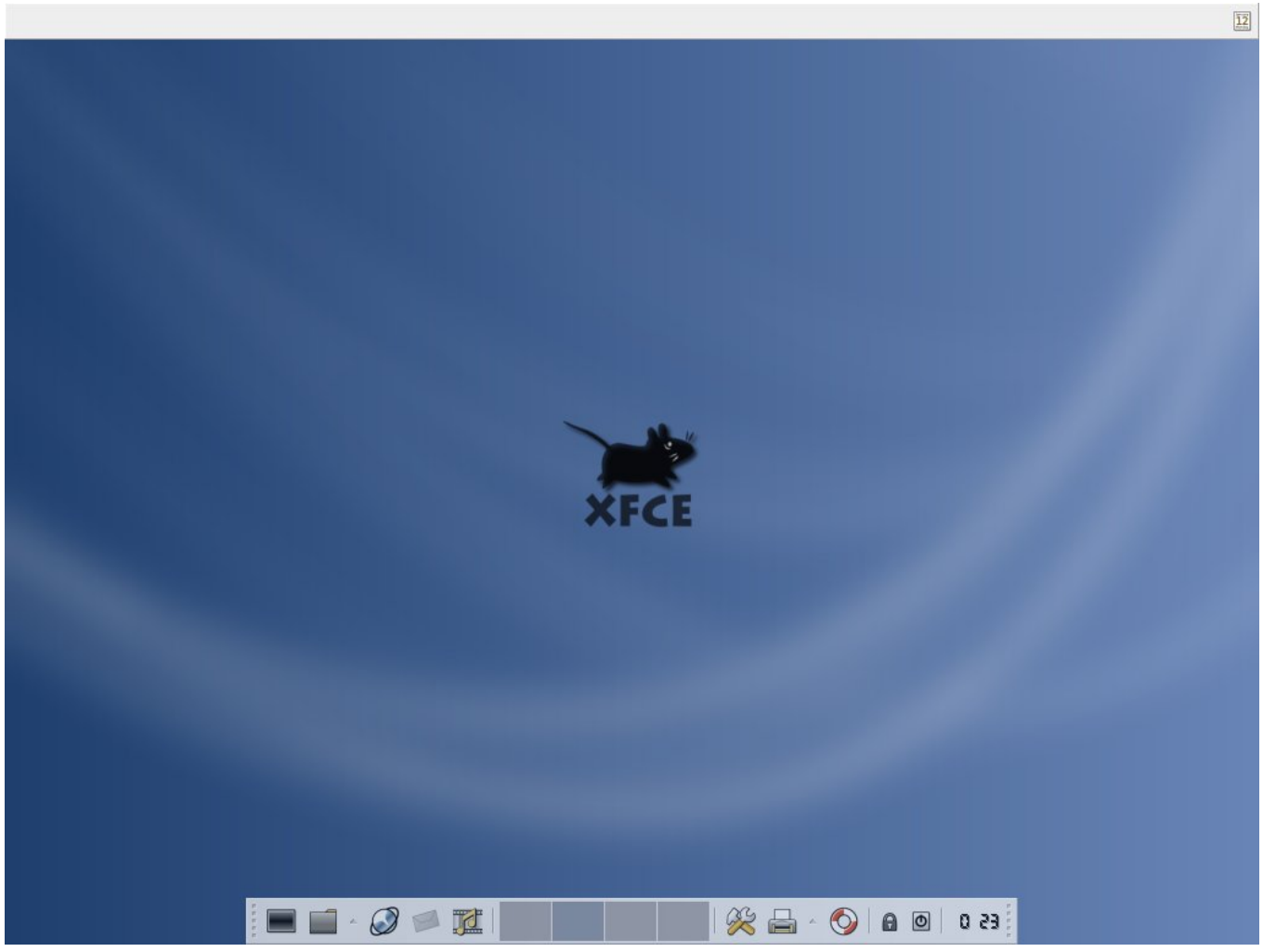

Figure 5: Xfce 4.0 (https://www.xfce.org/about/screenshots/)

Window Management

Stacking (floating)

Allows windows to overlap and be positioned freely, each on its own layer.

Tiling

Positions windows automatically on a single layer following a fixed algorithm.

Dynamic

Provides multiple, switchable tiling layouts.

(Dynamic Tiling) Window Managers: i3

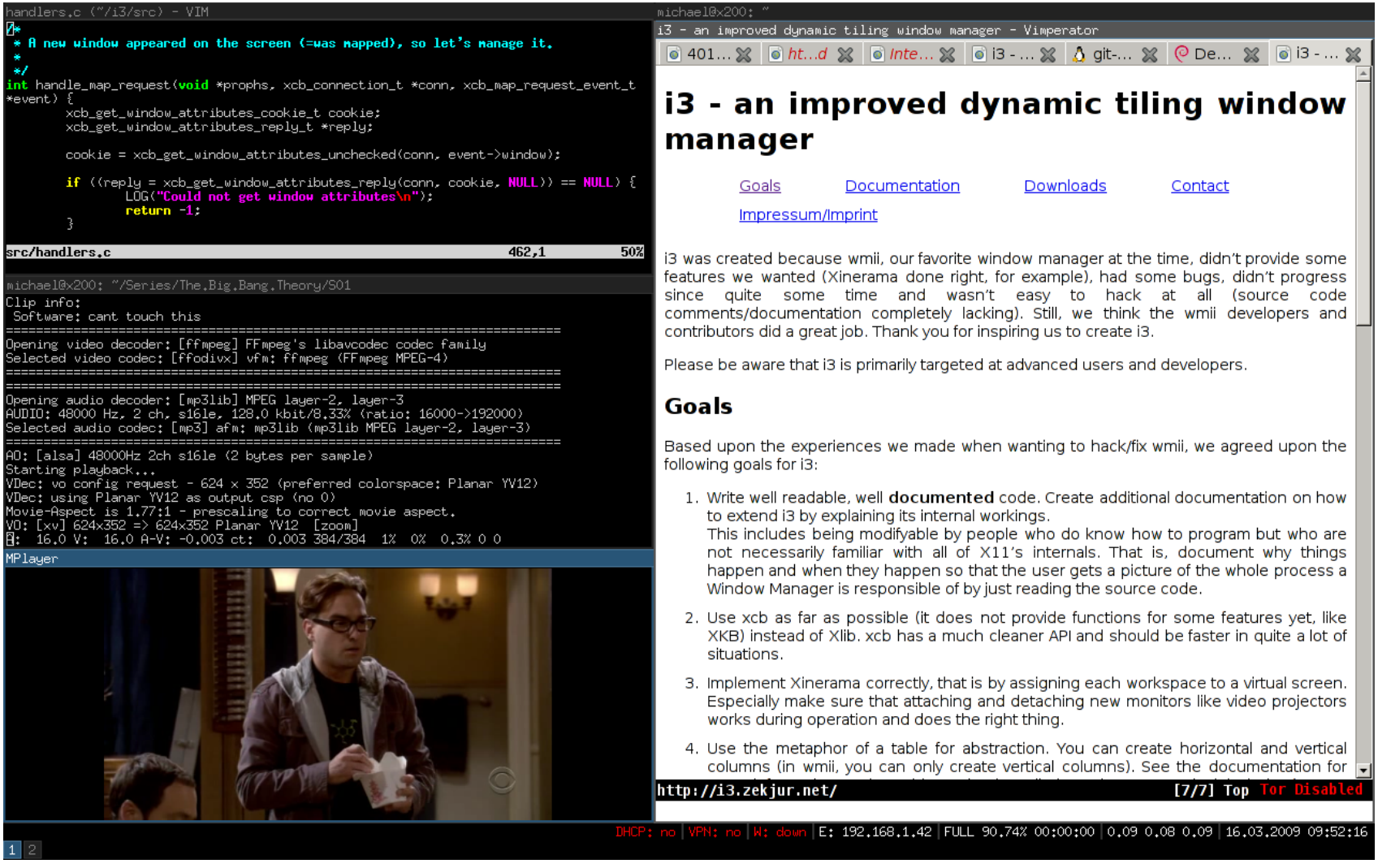

Figure 6: i3 (https://i3wm.org/#/screenshots/)

(Dynamic Tiling) Window Managers: awesome

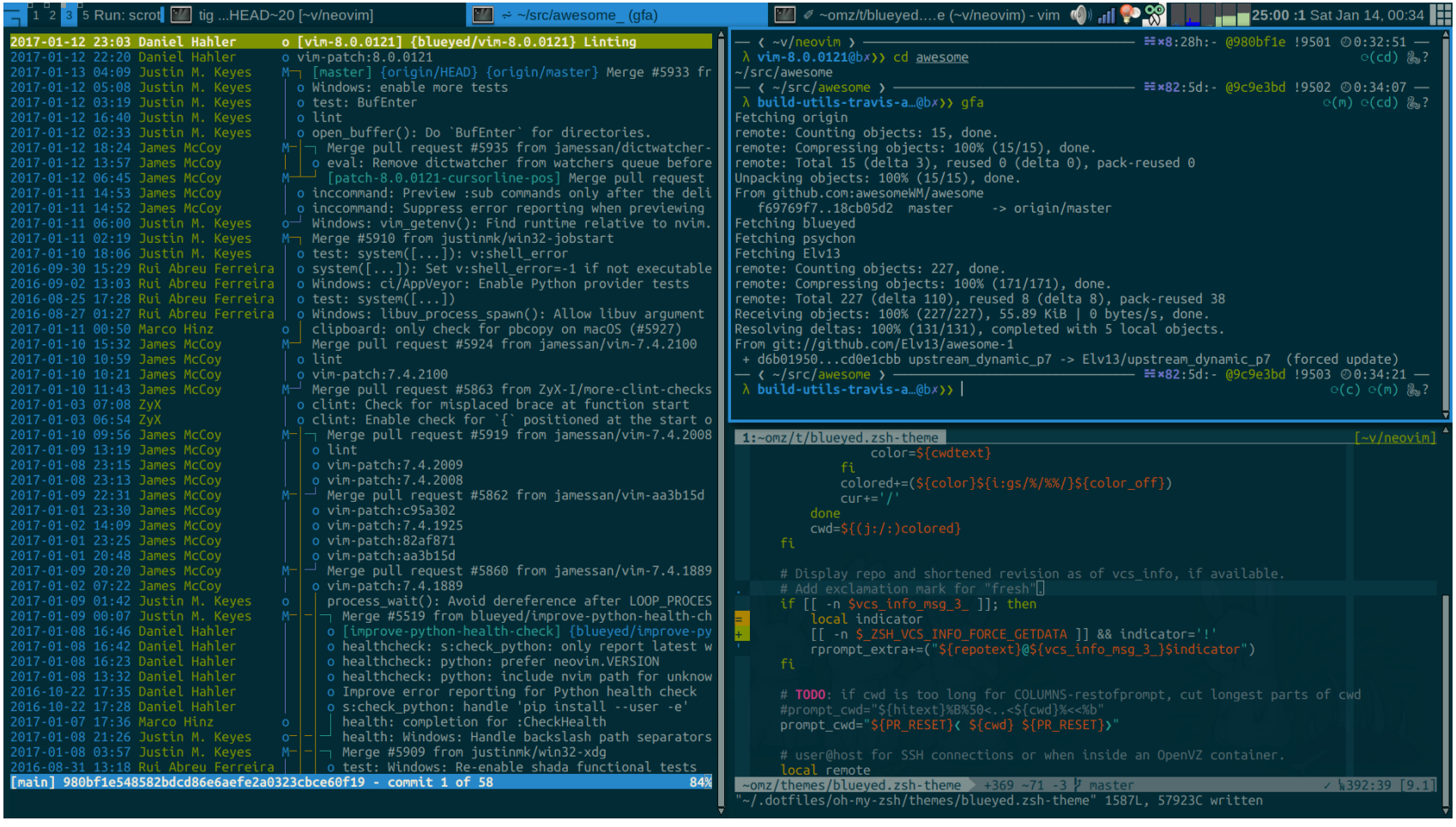

Figure 7: awesome (https://github.com/awesomeWM/awesome/issues/1395)

(Dynamic Tiling) Window Managers: XMonad

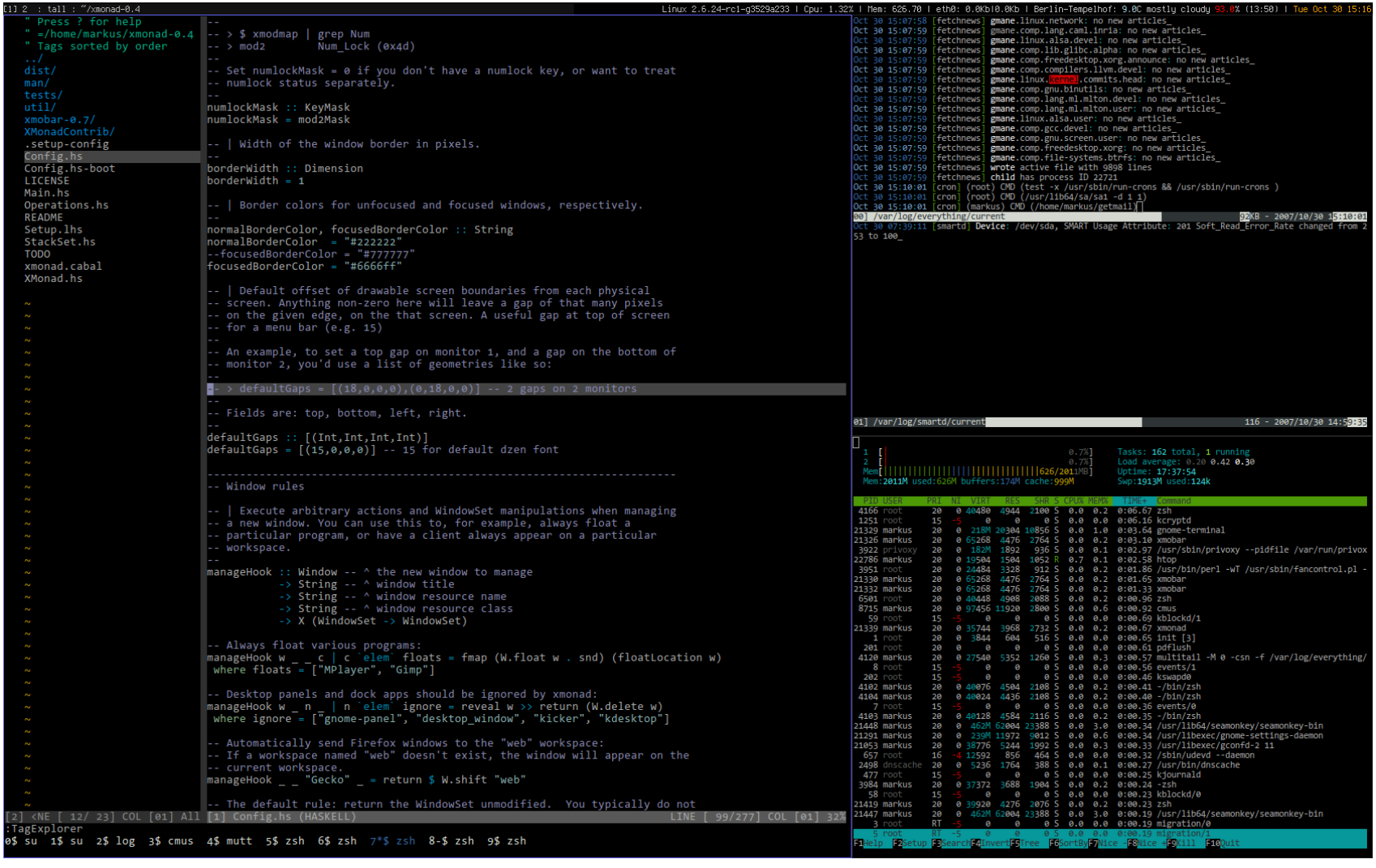

Figure 8: XMonad (https://wiki.haskell.org/Xmonad/Screenshots)

(Dynamic Tiling) Window Managers: dwm

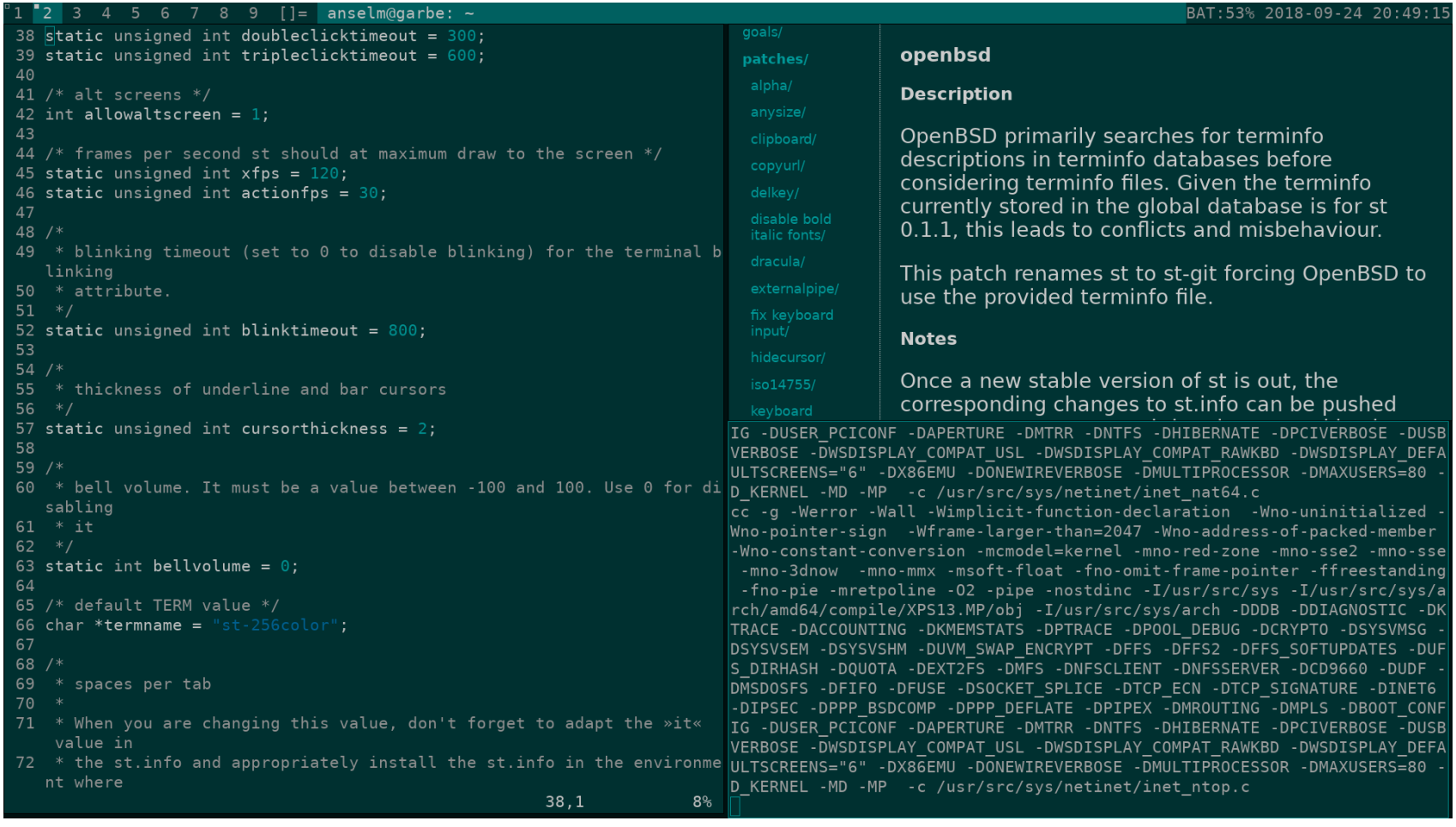

Figure 9: dwm (http://dwm.suckless.org/screenshots/)

tmux: Terminal Multiplexer

- Allows local and remote users to detach, reattach their sessions
- Automatically stores a session when the connection is lost
- Can be used as a dynamic window manager in the terminal

tmux Pane Navigation

Any `tmux` commands are preceded by `CTRL+B`.

`"`	Split pane into two, top and bottom
`%`	Split pane into two, left and right
`o`	Cycle between panes (or use cursor keys)
`CTRL+O`	Rotate panes
`CTRL+cursor`	Resize panes slightly
`ALT+cursor`	Resize panes more drastically
`z`	Toggle zoom to exclusive view
`ALT+1~5`	Layouts
`SPACE`	Cycle layouts
`x`	Kill pane (or type `exit` in shell)

tmux Window Navigation

`c`	Create new window
`0~9`	Move to window 0~9
`n`	Cycle between windows
`&`	Kill window
`:kill-window -a`	Kill all windows except the active one
`,` (comma)	Rename window

`tmux` Session Commands

`$`	Rename session
`d`	Detach client
`&` on last window	Kill `tmux` session (or enter `exit` in shell)

In the shell, `tmux kill-server` kills all sessions and closes `tmux`.

tmux Reattach and Session Name Options

`tmux attach`	Reattach the most recent lost or detached session
`tmux new -s name`	Start a new session called `name`
`tmux attach -t name`	Reattach a previously detached session called `name`
`tmux ls`	List current sessions (attached or detached)

Paste Buffer and Commands

`[`	Enter copy mode
`SPACE`	Start selection (no `CTRL+B` prefix)
`ENTER`	Copy selection (no `CTRL+B` prefix)
`q`	Quit mode (no `CTRL+B` prefix)
`]`	Paste from buffer
`:`	Open `tmux` command prompt

(NB in the default emacs mode, it's CTRL+SPACE, CTRL+W for start and copy selection.)

The command prompt allows such `tmux` commands as

```
rename-window newname
new-window -n '.bashrc' vim ~/.bashrc
```

These commands can also be invoked as `tmux` command-line options.

Configuration

`~/.tmux.conf`, e.g.

```
set -g prefix C-a
unbind C-b
bind-key C-a send-prefix
bind-key C-l clear-history
set -g status-style bg=colour24
set -g status-right ""
set-option -g set-titles on
set-option -g set-titles-string "#W"
```

bash Scripting

```
#!/bin/bash          # This line identifies the shell interpret-
                     # er; it should always be your first line.

echo "Hello world"   # The remainder of your file can simply be
                     # a sequence of things you'd otherwise writ
                     # on the command line.

                     # Hashes signal comments.
```

Make your file executable as follows:

```
chmod +x filename
```

Then run it with a leading path, as I haven't added any of your directories to `$PATH`:

```
./filename
```

A Simple bash Script

```
s3cmd sync ~/admin s3://backups/ --delete-removed -v
s3cmd sync ~/research s3://backups/ --delete-removed -v
s3cmd sync ~/teaching s3://backups/ --delete-removed -v
s3cmd sync ~/html s3://backups/ --delete-removed -v
```

- Just a sequence of commands that will work in all shells
- Thus no need to identify the shell interpreter

Variables

```
#!/bin/bash

string="Hello World!"
echo $string
```

- Set a text variable as `name="text"`, no spaces and no `$`
- Call it as `$name`, e.g. `echo $name`

Command Substitution

- Allows the output of a command to replace the command itself, so it can be processed more straightforwardly.
- Enclose the command in `$( … )` and assign it to a variable.

Command Substitution

- Allows the output of a command to replace the command itself, so it can be processed more straightforwardly.
- Enclose the command in `$( … )` and assign it to a variable.

```
#!/bin/bash

state=$(cat /sys/class/power_supply/AC/online)
echo $state
```

Integer Arithmetic: `let`

- Handles arithmetic evaluation
- No spaces unless you use quotes

```
#!/bin/bash

let addition=2+2
let subtraction=2-2
let multiplication="2 * 2"
x=10
y=2
let division=$x/$y
echo $addition          # etc.
```

Integer Arithmetic: Arithmetic Expansion

- Handles evaluation and command substitution
- Spaces permitted

```
#!/bin/bash

addition=$((2+2))
subtraction=$(( 2 - 2 ))
echo $(( 2 * 2 ))
x=10
y=2
division=$(( $x / $y ))
echo $addition          # etc.
```

A Simple Integer Arithmetic Script

```
#!/bin/bash

read -p "Enter two numbers separated by a space: " x y
ans=$(( x + y ))
echo "$x + $y = $ans"
```

Floating Point Arithmetic

`bash` itself can only handle integers. Floating-point solutions include:

`bc`	basic calculator
`calc`	has interactive mode
`awk`	a scripting language
`python`	a programming language

A Basic Floating-Point Script

```
#!/bin/bash

read -p "Enter any number: " x
root=$(echo "sqrt($x)" | bc -l )
echo "The square root of $x is $root."
```

Conditionals

```
#!/bin/bash

state=$(cat /sys/class/power_supply/AC/online)
if [ $state = "1" ]; then
    echo "AC"
    else echo "BAT"
fi
```

- Simple conditionals are signalled with `if` and closed with `fi`
- The condition is contained in brackets (double parentheses if arithmetic)
- The brackets **must** be spaced: they are a program (`test`)!
- Action to be undertaken is introduced by `then` or `else`
- The semicolon allows you to have multiple commands in one line

Conditionals

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

Long Form

```
if [ "$(whoami)" != 'root' ]; then
        echo "Permission denied."
        exit 1;
fi
```

Short Form

```
#!/bin/bash

[ "$(whoami)" != 'root' ] && ( echo "Denied"; exit 1 )
```

- && for “if successful”; || “if unsuccessful”
- `exit 0` reports success; `exit 1` error; `exit 2` fundamental issues