Technical Documentation

Andrei Suba
Computer Science Student,
West University of Timișoara,
Timișoara, Romania
Email: `andrei.suba00@e-uvt.ro`

Contents

1 Introduction **3**

2 Installation process **4**
- 2.1 Pre-installation 4
- 2.2 Installing the system 5
  - 2.2.1 installation_script.sh 5
  - 2.2.2 installation_script_part2.sh 6

3 Arch Linux **9**

4 Suckless software **10**
- 4.1 DWM 10
- 4.2 Dwmblocks 12
- 4.3 ST 13
- 4.4 Slock 14

5 Home directory structure **16**

6 Programs **17**

7 Troubleshooting **17**

8 Conclusion **18**

1 Introduction

This technical documentation was created for the Individual Project subject at my university and it's meant to describe my work for the past few months.
The project it's about an automatic script base configuration based on Arch Linux which follows the same design of minimalism with regards to its lightweight functioning and flexibility.
This linux distribution comes with a lightweight tiling window manager created by the Suckless Team named `dwm` and other tools like `dmenu`, `dwmblocks`, `st` and `slock`.
The implementation of their software and others will be explained thoroughly in the next sections.

Disclaimer: this work began a long time ago just as a personal project out of curiosity after going down the rabbit hole of minimalism setups. Shoutouts to Luke Smith!

2 Installation process

2.1 Pre-installation

There are two ways of installing the operating system and all configuration:

- **Getting the iso file**: directly burn the custom .iso file to an external media and boot from it. Run **installation_script.sh**:

  ```
  $: installation_script.sh
  ```

- **Any Arch Linux iso**: get the current iso of Arch Linux, burn it to an external media, booting from it and follow the below steps:

  1. First of all make sure that you have access to the internet.
     Follow the steps described here for help: Internet connection
  2. Initialize the keys: run the following commands

     ```
     #Initialize the local key
     $: pacman-key --init

     #Download the keyring
     $: pacman -Sy
     ```

  3. Install git:

     ```
     #Install git
     $: pacman --noconfirm -S git
     ```

  4. Clone the following repository: https://github.com/arghpy/individual_project:

     ```
     $: git clone https://github.com/arghpy/individual_project
     ```

  5. Change directory into it:

     ```
     $: cd individual_project
     ```

  6. Move both the `installation_script.sh` and `installation_script_part2.sh` into `/usr/local/bin/`:

     ```
     $: mv installation_script.sh /usr/local/bin/
     $: mv installation_script_part2.sh /usr/local/bin/
     ```

  7. Remove the cloned directory:

     ```
     $: rm -rf individual_project
     ```

  8. Run `installation_script.sh` and follow the steps:

     ```
     $: installation_script.sh
     ```

2.2 Installing the system

The installation process is done in two parts: first, the `installation_script.sh` is run, and after it finishes it displays a description on how the second part should be run (`installation_script_part2.sh`).

NOTE: the installation is not fully automated. It will still need interaction with the user, waiting in different parts for it's input/

2.2.1 installation_script.sh

After the user gets successfully past the 2.1 step and runs the `installation_script.sh` script, those will be the steps followed by the user:

1. **Internet**: first of all, the script will check for an internet connection by pinging the primary DNS of google (8.8.8.8). If it fails, the user will be prompted to check it's internet connection, point in which two options are possible to choose:
   - **Retry**: the pinging will be repeated
   - **nmtui**: for wifi connection. The steps that follow after launching it are simple to understand.
2. **Keys and configuring pacman**: it initializes a local key for pacman (package manager) and downloads all the signing keys from the Arch Linux developers. It also increasing the number of parallel downloads to 5.
3. **Choosing installation disks**: it check for the disks on the system on which the installation will be performed. The user will be prompted to choose the installation media from a menu which is displaying both the disk name (sda, sdb, nvme...) and their size, for the user to be able to differentiate them.

   <u>IMPORTANT NOTE</u>: **Please be careful when selecting the disk! After choosing it, all the data will be formated and the disk will be partitioned!**
4. **Partitioning**: the disk is going to be partitioned accordingly with their system boot mode: *BIOS* or *UEFI*. Following is the partition scheme for each boot mode:
   - **BIOS**: the disk will be partitioned on an MBR mode as follows:
     – `SWAP`: 4GB
     – `ROOT`: 30GB
     – `HOME`: space left
   - **UEFI**: the disk will be partitioned on an GPT mode as follows:
     – `BOOT`: 1GB
     – `ROOT`: 30GB
     – `HOME`: space left
5. **Formatting**: depending on the boot mode (*BIOS* or *UEFI*) the following formatting will be performed:
   - **BIOS**: the disk will be formatted as follows:

    - `SWAP`: [SWAP]
    - `ROOT`: EXT4
    - `HOME`: EXT4
  - **UEFI**: the disk will be formatted as follows:
    - `BOOT`: FAT32
    - `ROOT`: EXT4
    - `HOME`: EXT4

6. **Mounting the partitions**: depending on the boot mode (*BIOS* or *UEFI*) the following mounting point will be defined and partitions will be mounted:
   - **BIOS**: mounting points:
     - `SWAP`: the swap will be activated
     - `ROOT`: /mnt
     - `HOME`: /mnt/home
   - **UEFI**: mounting points:
     - `BOOT`: /mnt/boot
     - `ROOT`: /mnt
     - `HOME`: /mnt/home
7. **Installing packages**: the script will download packages.csv file and based on it, it will install all packages that are from the repository into the new system.
8. **Automatically mounting partitions at boot**: the script will generate the fstab file (File System Table) for the new system such that the newly created system will be able to recognize it's partitions and mount them.
9. **Finish**: after the previous step, the script will print a description that exemplifies what the user should be doing next (how to run `installation_script_part2.sh` based on the boot mode automatically identified before - *BIOS* or *UEFI*).
   After the description is displayed, the script will change the root directory to the new system.

2.2.2 installation_script_part2.sh

This part continues immediately after the `installation_script.sh` finished. The user should already be located in the newly installed system, viewing above the prompt the following text:

```
Now entering the system.

The boot mode is: UEFI.

To continue with the installation process execute the script
installation_script_part2.sh specifying the mode.
Example:

$: installation_script_part2.sh BIOS

$: installation_script_part2.sh UEFI
```

NOTE: in this example the boot mod is *UEFI*, but depending on the system this value can also be *BIOS*.

Below are the next steps done by the installation script:

1. **Keys initialization**: during this step, the following commands are executed in order to create a local pacman key and register the keys of the developers:

   ```
   pacman-key --init
   pacman -Sy
   pacman-key --populate archlinux
   ```

2. **Setting the time**: the time is set for Europe/Bucharest by creating a symbolic link from `/usr/share/zoneinfo/Europe/Bucharest` to `/etc/localtime`. In this step the hardware clock is also synced with the system time.

3. **Language**: the language is set to *English US*.

4. **Hostname**: the user will be prompted to type the hostname of the system (name of the station , viewed by the local network).

5. **User**: the user will be prompted to type the name of the local user on the system. In this step the script will create a local user based on the name typed before. After the user is created, the script will ask for the password for the newly created user. Please note that the password will need to be typed twice.

6. **GRUB**: *GRUB* is installed in this step and the system will be set to boot using it.

   NOTE: this step configures GRUB based on the boot mode. That's why the installation script in this part needs to be run with an argument.
   Be careful to the argument which is provided to the script.*Double-check!*

7. **Services**: the *Network Manager* service is started and enabled in order for the new system to have access to internet.

8. **Default directories**: the file `/etc/xdg/user-dirs.conf` is modified such that upon creating any new users, just the contents of `/etc/skel` will be copied into the home directory.

9. **Setting the home directory**: additional files are added to the user's home directory, files which are copied from the GitHub repository: local_config.

   Those files contain the configuration that will be binded with programs installed in the first part of the installation, constructing the environment.

10. **Compiling Suckless software**: the script goes in the directory `/.local/src`, searches for the directories `dwm, dwmblocks, dmenu, st` and `slock`, goes into each of them and builds them.

11. **AUR software**: `AUR` stands Arch User Repository. In this step, the script installs additional software that can be found only in AUR. This

installation is done using `yay`: a command-line tool used for managing software from AUR.

12. **Login Manager**: the `lightdm` login manager is installed and enabled. It's lightweight, simple and easy to configure. It's also a graphical login manager.

13. **EarlyOOM**: in this step the script enables `earlyoom` which is a service used to stop an OutOfMemory event by monitoring the RAM usage.

14. **Finish**: this is the final step of the installation. The user will see the message:

```
Installation finished.
Type ‘exit‘ to get out of chroot and after that type
‘shutdown now‘, take out the installation media and boot into
the new system.
```

After the steps described are performed, all of the installation finished, and the system is ready.

3 Arch Linux

In conformity with the Arch development team's view,
"Arch Linux is an independently developed, x86-64 general-purpose GNU/Linux distribution that strives to provide the latest stable versions of most software by following a rolling-release model. The default installation is a minimal base system, configured by the user to only add what is purposely required."

This minimalist distribution of linux is highly popular for its customizability and was developed by having the following principles in mind: simplicity, modernity, pragmatism, user centrality and versatility.
Because of the above princpiles many users choose to use this distribution as their main distribution, myself included. Given the power to the user and being minimal, it provided me and many others the means to customize it to fit our personal needs and beliefs.

4 Suckless software

The suckless.org community is a group of developers who create and maintain a collection of minimalist and efficient software programs. The community was founded by the developers of the DWM window manager, and has since grown to include a variety of software projects that adhere to the suckless philosophy. The community's goal is to create simple and efficient software that is easy to understand and customize, while avoiding unnecessary features or bloat.

The suckless.org community is known for its focus on minimalism and simplicity, and its software projects are often designed to do one thing well and avoid unnecessary features. The community also emphasizes the use of clean and readable code, and encourages users to customize and modify the software to suit their own needs.

Some of the most popular software projects developed by the suckless.org community include the DWM and ST window managers, the dmenu application launcher, and the sutils collection of command-line utilities. All of these projects are available as free and open-source software, and are widely used by users of Linux and other Unix-like operating systems.

DWM, Dwmblocks and ST as well as other programs compiled from source can be found in the user's home directory under `.local/src/`.

```
.local/src
.local/src/yay
.local/src/dmenu
.local/src/dwmblocks
.local/src/st
.local/src/dwm
.local/src/slock
```

4.1 DWM

DWM is a dynamic window manager for the X Window System. It is a lightweight and minimalistic window manager that is designed to be fast and efficient, and is often used in conjunction with a tiling window manager. DWM is written in C and is highly customizable, allowing users to configure and modify its behavior to suit their preferences.

DWM is known for its simplicity and efficiency, and is often used by users who want a fast and lightweight window manager that can be easily customized. Some of the key features of DWM include:

- **Tiling layout**: DWM automatically arranges windows in a tiled layout, which can make it easier to manage multiple windows and improve productivity.
- **Customizable appearance**: allows users to customize its appearance, including the colors, font, and layout of the window decorations.
- **Keyboard-based navigation**: can be controlled entirely from the keyboard, which can make it faster and more efficient to use.

- **Extensibility**: is highly customizable, and users can easily add new features or modify its behavior by writing their own patches or scripts.

The configuration file for DWM is found under `.local/src/dwm` with the name `config.h`. The patches applied on the original DWM software are found in `.local/src/dwm/patches` and include:

- **fullgaps**: used to implement gaps between windows within the same tag
- **colorbar**: for adding colors to the Dwmblocks status bar
- **statuscmd**: it is used to create a link between DWM and Dwmblocks
- **systray**: added in order for applets to be displayed (e.g.: `nm-applet`)

The `config.h` file contains the following personal additions:

Font size:

```
static const char *fonts[]       = { "monospace:size=14" };
static const char dmenufont[]    = "monospace:size=14";
```

Colors in hex format:

```
static const char col_black[]    = "#000000";
static const char col_gray1[]    = "#222222";
static const char col_gray2[]    = "#444444";
static const char col_gray3[]    = "#bbbbbb";
static const char col_gray4[]    = "#eeeeee";
static const char col_gray5[]    = "#3f4e4f";
static const char col_cyan[]     = "#005577";
static const char col_cyan2[]    = "#a5c9ca";
static const char col_black1[]   = "#2c3333";
static const char col_black2[]   = "#2c3639";
static const char col_BlGreen[] = "#3f4e4f";
static const char col_violet[]   = "#51557e";
static const char col_blue1[]    = "#395b64";
static const char col_blue2[]    = "#00a8cc";
static const char col_blue3[]    = "#151965";
static const char col_blue4[]    = "#007880";
static const char col_blue5[]    = "#2d6e7e";
static const char col_blue6[]    = "#1597bb";
static const char col_white2[]   = "#e7f6f2";
```

Which are to be added in:

```
static const char *colors[][3]   = {
       /*             fg        bg        border   */
       /*[SchemeNorm] = { col_gray3, col_gray1, col_gray2 }, ORIGINAL */
       [SchemeNorm] = { col_white2, col_BlackGreen, col_gray2 },
       /*[SchemeSel] = { col_gray4, col_cyan, col_cyan }, ORIGINAL */
       [SchemeSel] = { col_white2, col_BlackGreen, col_cyan },
       [SchemeStatus] = { col_white2, col_gray1, "#000000" },
       [SchemeTagsSel] = { col_black, col_cyan2, "#000000" },
       [SchemeTagsNorm] = { col_gray3, col_gray1, "#000000" },
       [SchemeInfoSel] = { col_gray4, col_cyan, "#000000" },
       [SchemeInfoNorm] = { col_gray3, col_gray1, "#000000" },

};
```

Number of tags and name:

```
static const char *tags[] = { "1", "2", "3", "4", "5" };
```

Calling programs and local scripts:

```
static const char *termcmd[] = { "st", NULL };
static const char *firefox[] = { "/usr/bin/firefox", NULL};
static const char *lynx[] = {
    "/home/$USER/.local/bin/dwm_scripts/start_lynx", NULL};
static const char *thunar[] = { "/usr/bin/thunar", NULL};
static const char *lf[] = {
    "/home/$USER/.local/bin/dwm_scripts/start_lf", NULL};
static const char *slock[] = { "slock", NULL};
```

Making key-bindings:

```
static const Key keys[] = {
    /* modifier                 key        function       argument */
    { MODKEY,                   XK_d,      spawn,         {.v = dmenucmd } },
    { MODKEY,                   XK_Return, spawn,         {.v = termcmd } },
    { MODKEY,                   XK_w,      spawn,         {.v = firefox } },
    { MODKEY|ShiftMask,         XK_w,      spawn,         {.v = lynx } },
    { MODKEY|ShiftMask,         XK_o,      spawn,         {.v = thunar } },
    { MODKEY,                   XK_o,      spawn,         {.v = lf } },
    { MODKEY|ShiftMask,         XK_l,      spawn,         {.v = slock } },
   ...
```

4.2 Dwmblocks

Dwmblocks is a status bar program for the DWM window manager. It is a simple and lightweight program that displays various system information, such as the date and time, CPU and memory usage, and network activity, in a customizable status bar at the top of the screen. Dwmblocks is written in C and is highly configurable, allowing users to customize the information that is displayed and the appearance of the status bar.

Dwmblocks is designed to work seamlessly with DWM, and is often used by users who want to display system information in the status bar of their DWM-based desktop environment. Some of the key features of Dwmblocks include:

- **Customizable display**: Dwmblocks allows users to choose which system information to display in the status bar, and to customize the layout and appearance of the status bar.
- **Multiple data sources**: can display information from a variety of sources, including the system clock, CPU and memory usage, and network activity.
- **Mouse-based interaction**: allows users to interact with some of the displayed information using the mouse, such as by clicking on the clock to open the calendar.
- **Extensibility**: Dwmblocks can be easily extended by writing custom scripts to display additional information or provide new features.

The configuration file for Dwmblocks is found under `.local/src/dwmblocks` with the name `blocks.h`. The patches applied on the original DWM software are found in `.local/src/dwmblocks/patches` and include:

- **statuscmd**: used to provide clickability and better integration with DWM

The `blocks.h` file contains the scripts whose output is to be displayed in the status bar. The scripts are located in the users's home directory under `.local/bin/dwmblocks_scripts/`. When added in `blocks.h` file, they need to be added in the form:

```
/*Icon*/   /*Command*/   /*Update Interval*/  /*Update Signal*/
{"",       "disk_usage",         30,                  16},
```

Where **Command** is the script, **Update Interval** is the amount of time in seconds at which the system sends a signal to the already running process **dwmblocks** to update, based on the signal from **Update Signal** + 44. An example of how would this look like:

```
kill -47 $(pidof dwmblocks)
```

In order to add click functionality to your scripts, the following needs to be added:

- **DWM**: click buttons need to be defined

  ```
  static const Button buttons[] = {
      { ClkStatusText, 0,         Button1,  sigstatusbar,  {.i = 1} },
      { ClkStatusText, 0,         Button2,  sigstatusbar,  {.i = 2} },
      { ClkStatusText, 0,         Button3,  sigstatusbar,  {.i = 3} },
      { ClkStatusText, ShiftMask, Button1,  sigstatusbar,  {.i = 6} },
      ...
  }
  ```

  `Button1` represents Left Click
  `Button2` represents Wheel Click
  `Button3` represents Right Click

- **Script**: in the scripts, there needs to be defined a 'case' function to check for the buttons on which and implementation is desired:

  ```
     case $BUTTON in
         1) dunstify "$RESULT" ;;
         3) st -e htop ;;
         6) st -e vim $(which $0) ;;
     esac
  ```

  In the above example, `6)` is going to represent `Shift + Left Click` which is going to open in `vim` the script.

4.3 ST

ST (short for "suckless terminal") is a simple and minimalistic terminal emulator. ST is written in C and is highly configurable, allowing users to customize its appearance and behavior to suit their preferences.

ST is designed to be a fast and efficient terminal emulator, with a focus on simplicity and minimalism. Some of the key features of ST include:

- **Customizable appearance**: ST allows users to customize the colors, font, and size of the terminal, as well as the window decorations.
- **Keyboard-based navigation**: can be controlled entirely from the keyboard, which can make it faster and more efficient to use.
- **Unicode support**: supports the Unicode character set, allowing it to display a wide range of characters and symbols from various languages and scripts.
- **Extensibility**: is highly customizable, and users can easily add new features or modify its behavior by writing their own patches or scripts.

The configuration file for ST is found under `.local/src/st` with the name `config.h`. The patches applied on the original ST software are found in `.local/src/st/patches` and include:

- **anysize**: added in order to work correctly with the `gaps` patch from DWM (without it gaps may differ in size when multiple windows are open)
- **alpha**: to add opacity/transperency
- **scrollback-ringbuffer**: it is used to scrollback in the terminal

The `config.h` file contains the following personal additions:

Font size:

```
static char *font = "Liberation
    Mono:pixelsize=18:antialias=true:autohint=true";
```

Shell:

```
static char *shell = "/bin/sh"
```

The opacity (it's a range between 0 and 1):

```
float alpha = 0.9;
```

4.4 Slock

Slock is a simple, light-weight lock manager for Linux and Unix-like operating systems. Slock is designed to be easy to use and customize, and it has a command-line interface that allows users to quickly lock and unlock their screens. It is a popular choice among users who value simplicity and speed in their software.

The configuration file for Slock is found under `.local/src/slock` with the name `config.h`.
The `config.h` needs those modifications in order to function:

```
static const char *user = "user";
static const char *group = "wheel";
```

The value **user** needs to be modified accordingly with the local user from the system.
The group **wheel** represents the group with root privileages.

Important Note:

In each of the suckless programs, there are two important configuration files: `config.h` and `config.def.h`. Any modifications should be done on `config.h`. After the modifications have been applied, the user (with **root** privileages) needs to run the following command:

```
sudo make clean install
```

For more information about the build process check the `Makefile`.

5 Home directory structure

The user's home directory should have the following structure:

```
./.vim
./.vim/vimrc
./.local
./.local/src
./.local/bin
./.ssh
./.zshrc
./.zsh_history
./pictures
./pictures/wallpapers
./.profile
./.config
./.config/dunst
./.config/user_guide.pdf
./.config/icons
./.config/lf
./.xinitrc
```

- Under **.vim** should be all user configuration for the vim editor, including any additional plugins.
- **.local/bin**: should store user scripts and programs
- **.local/src**: should store programs installed by source and compiled
- **.ssh**: all user specific configuration for `ssh`
- **.config**: should contain all user configuration about system-wide and local programs

The **pictures/wallpapers** directory is a personal addition created for pictures that are to be displayed as the desktop's wallpaper.

The files **.zshrc** and **.profile** are usual for a system in which a login is required for a user that has as the default shell `/bin/zsh`. The **.xinitrc** file is specific for the **X Window** and is used to start programs at login upon executing `startx` (which in this case is automatically executed in `.zshrc`).

6 Programs

All installed programs are listed and described in packages.csv file.

They were chosen to reflect the minimalist and lightweight style of the linux distribution. Their purpose is to increase efficiency by using mostly only the keyboard.
Given their simple and lightweight design, stability is also one of the reasons why they were chosen.

7 Troubleshooting

In case that the keys cannot be initialized correctly, the following should be done:

```
#Open for editing the file /etc/pacman.conf (with nano or vim)
$: vim /etc/pacman.conf


#Search for the line with ‘ParallelDownloads‘ and modify it
#like this:
ParallelDownloads = 5


#Search for the line with ‘SigLevel‘ and modify it like this:
SigLevel = Never

#Save the file

#Initialize the key:
$: pacman-key --init

#Download the current keys for arch
$: pacman --noconfirm -Sy archlinux-keyring
```

After the above steps are performed, edit the file again as it was, except for the parallel downloads if this interests you.

8 Conclusion

The project successfully combined the minimalist philosophy of the suckless community with the flexible and powerful Arch Linux operating system. By carefully selecting and installing a range of software from the suckless community and other software that shares the same philosophy, a custom Linux distribution emerged that prioritized simplicity, efficiency, and customization. This project serves as a confirmation to the value of minimalist and efficient design.

This project was a rewarding and satisfying experience for me because it integrates different values that I care for like minimalism and open-source software. It challenged me to think creatively and critically about different approaches to accomplish this result.