

# **A Farewell to Feliz**

by

**Liz Mills**

The Developer Behind

The Quick and Easy Arch Linux Installer

Bows Out

A Farewell to Feliz - The Quick and Easy Arch Installer  
Copyright © 2016 and 2017 by Elizabeth Audrey Mills  
Published by Elizabeth Audrey Mills

Elizabeth Audrey Mills asserts her right to be identified as author of this work in accordance with the Copyright, Designs and Patents Act 1988. All rights reserved.

Without limiting the rights under copyright reserved above, you may share this ebook with your friends provided it remains in its complete original form, including the cover and this copyright page, and no charge is made. This ebook must not be sold or distributed for commercial purposes by any person or persons other than the seller or sellers authorized by the above author and publisher.

~~~~~

Written using [LibreOffice](#) by [The Document Foundation](#)

Email: [elizabeth@itsliz.net](mailto:elizabeth@itsliz.net)

Website: [www.itsliz.net](http://www.itsliz.net)

## Introduction

Why am I stepping back from developing Feliz? Well, I'm not getting any younger, and although I know more now than ever before in my life, my 73-year-old brain is finding it harder to process all that knowledge. Coding projects are taking longer to complete and I find that they contain bugs that I am unable to eradicate. I have to face the fact that, before long, I will be unable to continue to maintain Feliz and the other projects I have started. The time has come for someone younger to take over, to clone Feliz and make their own version and share it with the world. The Feliz scripts will remain as they are, to serve as a source for anyone else who fancies giving it a go.

So this book is intended as a handover document to anyone who may think it worthwhile continuing my work.

Essentially, it is a guide to the Feliz scripts for installing Arch Linux, although most of the principles can also be applied to my other projects. It is not a guide to scripting, you'll have to sort that part out for yourself. If you need a good guide to programming the GNU/Linux shell, I recommend 'Classic Shell Scripting' by Arnold Robbins & Nelson H. F. Beebe published by O'Reilly, 'Definitive Guide to sed' by Daniel A Goldman, and the invaluable 'Pocket Reference' series of concise reference books also from O'Reilly. However, if you have some basic experience in any programming language, and understand such concepts as variables and functions, then you will probably have no trouble with the code in Feliz.

The purpose of this little book is to outline the development and structure of the Feliz project, emphasising 'good practice', so that anyone who reads the scripts will be able to follow them. I will also share the techniques that help with the development process, right up to a guide on how I make the Feliz iso.

Feliz is free and open-source, meaning that you can use it and adapt it to your needs, or even include parts of it in your own scripts, subject to the terms of the GNU General License, a copy of which is included with this book. You can find all the code of all the Feliz modules, and my other small contributions to the GNU/Linux world, in the angeltoast repositories at Github: <https://github.com/angeltoast>

## Table of Contents

|                                                 |    |
|-------------------------------------------------|----|
| 1 - About Feliz.....                            | 1  |
| 2 - About Arch Linux.....                       | 3  |
| 3 - Using Feliz to Install Arch Linux.....      | 4  |
| 4 – Developing Feliz - Directory Structure..... | 7  |
| 5 - The Feliz Initialising Script.....          | 10 |
| 6 – Feliz Development - File Structure.....     | 11 |
| 7 – Developing Feliz - Listgen.....             | 13 |
| 8 - How Feliz Became Multilingual.....          | 23 |
| 9 - Thank You Linus Torvalds For Git.....       | 31 |
| 10 – Using and Building the Feliz ISO.....      | 33 |
| return 0.....                                   | 36 |
| done }.....                                     | 36 |

## 1 - About Feliz

Feliz is a set of scripts that utilise the tools that are part of the BASH shell to automate the installation of Arch Linux. Feliz is freely shared in the spirit of the community, subject only to the terms of the GNU General Public License.

When I first wrote the originals (known at that time as Achoo) it was at the start of a process of learning shell scripting, and it was for my own use – I was simply turning the Arch Wiki Beginners' Guide into a script with my preferences coded in, as a way to speed up the process. However, once it was working and I started to share it, people asked for a more general version, that could work in their own situations. Although it was perfectly possible for anyone to edit the scripts to suit their needs, not everyone has those skills, so it was clearly necessary for me to include options for setting such things as location and the partitioning of the device. Also, friends asked for added facilities, and so more and more functions were added to meet their requests. Before I knew it, Achoo had grown into a full-blown independent installer.

At that time, there were two other installers available for Arch Linux:

- Evo/Lution, by Jeff Storey and Carl Duff
- the AUI scripts by HelmuthDU.

I had used both, and they were excellent, but the AUI scripts were apparently no longer maintained, and it looked as though the same was happening to Evo/Lution. It seemed a good time to create a new alternative.

Achoo had one distinction, which Feliz has inherited - the user sets all the data before installation begins, and once that is done, installation is then fully automatic, with no further intervention by the user. This makes it very quick, typically installing a full system, with desktop environment and other software, in around 15 minutes, and a base Arch Linux installation in as little as 5 minutes on a fast internet connection.

Why was it called Achoo? Well, when I first started sharing my script, it was rather crude and basic, and I wanted to give it a name that reflected how fast it is to use, compared with doing it by hand. I had a head-cold at the time, and it seemed to me that my project was a bit like a sneeze - quick and dirty - so that's why I named it Achoo!

Vanity then took over. This was at the time we heard the sad news of the passing of Ian Murdock, the creator of the Debian distribution. Ian's name is immortalised in his distribution, and I decided that I wanted my installer to carry my name, too. So Achoo became Feliz - which means 'happy' in the Spanish and Portuguese languages - and Liz is a shortened version of my first name.

Feliz is distributed as an iso, which contains a copy of the Arch Linux live environment, from which the installation will take place, and a small script that initialises the Feliz system. Once the Arch Linux live session has started, the Feliz initialisation script checks the internet connection then downloads the latest copies of all the necessary scripts and files to complete the installation.

My old-fashioned style of coding probably betrays my age, and my roots in older database programming systems (Clarion, in my case), but this has the advantage that it is easy for anyone to follow.

Despite the apparent complexity of the Feliz scripts, I do not present this book claiming to be any kind of expert in scripting (or, indeed in Arch Linux). I am still very much a beginner, and owe a great deal to the people who have gone before me. In particular, I learnt much from reading the Evo/Lution (now called Architect) scripts - written by Carl Duff - which serve the same purpose as Feliz, though working in a very different way. Some of the code in Feliz has been adapted from Carl's excellent work, and Carl has since become a friend. Architect is the most comprehensive installer for Arch Linux, and is close to the recommended way of installing Arch as detailed in the Arch Wiki. Feliz has taken a different course, aiming to take care of as many of the details as possible, to help a beginner to get their first Arch system up and running. With the confidence gained from their first success, I hope they will then try Architect, and then a fully unaided installation following the Wiki.

Since the birth of Feliz, another excellent installer has appeared - Arch Linux Anywhere (subsequently renamed Anarchy) - and I am happy to say that its creator, Dylan James Schacht is also a good friend. I am also delighted to report that the scripts that started it all, the AUI Scripts, are still maintained by their creator.

And Feliz has advanced, too, particularly in its inclusion of a range of applications from the Arch repositories, offering desktop environments and dozens of other software packages that can be installed with your new Arch system.

Feliz is proudly cli ('command-line interface' - meaning that it presents white text on a black background). There are no mouse controls and no graphics, just simple menu choices and data-entry fields. And Feliz does not use Dialog, or any other semi-graphical displays, although it does feature interactive menus – these are hand-coded in Feliz, via the Listgen functions (described in this book). It feels more like the manual installation process, as you would do it with the Arch Beginners' Guide, and that is deliberate, because after installation you will be using the terminal a lot – Arch makes sure of that. Nevertheless, it is a friendly, helpful environment, and you should not feel intimidated by it.

## 2 - About Arch Linux

GNU/Linux, sometimes casually referred to as just 'Linux', is more than an operating system, it is a philosophy and a worldwide community. Each of the many distributions is developed and maintained by teams of enthusiasts, most of them volunteers, and supported by users who are, for the most part, passionate about the concept of freedom in computer software. This freedom is at the heart of the GNU/Linux ethos ... freedom to share, change, copy and add to the vast repository of programs that is GNU/Linux. Because of the organic nature of the system and the passion of its contributors, most software in the GNU/Linux family is not only free in the sense of freedom, but also free to obtain - there is nothing to pay.

Arch Linux is just one of these distributions, independent of any others, with a devoted community of followers. Its website includes an extensive wiki, covering everything from a beginners' installation guide to information about pretty much all the software you are likely to run. One feature of Arch is the vast library of applications that have been compiled and shared in the AUR (Arch User Repository), as well as the extensive official repositories.

Arch aims to follow the principle of 'keep it simple.' In this context, 'simple' does not mean 'easy', but rather 'uncomplicated', and Arch has a reputation for being difficult to use. Certainly, it is harder for newcomers than many of the popular distributions, such as Linux Mint or Ubuntu, or even Debian; but what you gain is a system that is exactly configured to your requirements

As a developer, specifically a solo developer of an independent Arch installer, I have a love-hate relationship with Arch Linux ... it would be wrong of me to say otherwise. The Wiki, though extensive (or, perhaps, because of it) can sometimes leave the head spinning. However, the answer is usually in there somewhere, and there is a sense of satisfaction in reaching the end of the journey and looking back at the obstacles overcome.

Feliz is an anachronism - an easy-to-use installer for a distribution that requires considerable technical ability and a 'can do' mindset to maintain. The Arch community will never recommend Feliz, because it goes against all the principles on which Arch exists. My excuse for Feliz is that it can help to open the door to a distribution that may otherwise be intimidating for first-time users, and that once that new user has seen how the system works, they can then learn about all the nuts and bolts.

### 3 - Using Feliz to Install Arch Linux

To get Feliz, just go to:

<http://sourceforge.net/projects/feliz/>

Click on the big green button that says “Download” and, depending on the speed of your internet service, the iso will be download to your computer in somewhere between six and sixty minutes. Everybody says this, and you're probably bored with hearing it, but before you use the iso, check the checksum. A bad download can cause all kinds of problems, so it is wise to make sure you have a good copy. The md5 and sha1 checksums for all releases of Feliz or Achoo can be found under the ( i ) icon beside each release on the download page.

If you just want to test Feliz in Virtualbox or another virtual system, just set up a new virtual machine with the Feliz iso in its virtual CD drive, assign the recommended ram and virtual hard-disk allocation (or more) and you are ready to roll.

If you plan to install on real hardware, you will first need to burn the iso image to a USB flashdrive or a CD, using whatever software you prefer, or which your current operating system offers (most of the user-friendly distros include one).

When you start up the installer, choose the right architecture for your system, and wait while Arch loads - it should happen quickly.

After a few seconds, the installer will inform you that it is testing for an internet connection. This is necessary, as all the scripts and files for the installation will be downloaded as needed. If no internet is available, the installer will display a message saying that it cannot continue, then it will abandon the installation. It can use either wifi or ethernet cable connection, but a slow wifi can cause the installation to fail. If possible, use a wired connection to install (your system should still work by wifi afterwards).

Feliz keeps you informed and advised at every stage, whilst trying to avoid filling the screen with too much text. If you need more guidance at any stage, it is a good idea to refer to the Beginners' Installation Guide in the Arch Wiki ...

[https://wiki.archlinux.org/index.php/Beginners%27\\_guide](https://wiki.archlinux.org/index.php/Beginners%27_guide)

The Feliz scripts closely follow that guide (although some stages are simplified by making certain generalisations) so you should be able to find out more about any of the steps just by reading the Wiki.

Most options are offered either through interactive menus or (if there are many options) as numbered lists - you simply enter the number of your selection.

#### **Location**

The user only needs to provide the minimum information required by Arch Linux to carry out an installation. These relate mainly to location (for setting the system clock), language (both for the data-entry stage and for the installed system – which can be different, if desired), keyboard layout, plus a few other options. First you choose your world zone, then you get to set the fonts and language of your installation, based on your locale - a combination of the language you wish to use and the country in which you are located. Mostly it is a case of picking from a list of suggestions.



## **Keyboard**

The next step is to define your keyboard layout. Feliz will offer a list of suggestions, but if yours is not on the list, you have the option of entering the name of your keyboard layout if you know it. If you can't find your keyboard, the installer allows you to 'Retry' as often as necessary until you find the right one.

## **Extras**

Now we arrive at one of Feliz great distinctions. You will enter a virtual shop where you can choose from lists of some of the free goodies that are part of the Arch Linux official repositories. Wander among the departments and choose as many as you like. Mix and match all you like from such things as DEs (desktop environments) web browsers, programming tools, office software and various accessories. This step is optional, so that you have complete control over your new system. You can choose to either have a basic Arch installation, without a desktop or any other graphics settings (some experienced Archers like it this way, so they can build everything exactly as they want it) or the installer can install your choice of desktop and other software. There is even a simple, pre-configured 'FelizOB' option, which will install a complete system based on Openbox. You choose – it is not for me to tell you what to do here. If you just want base Arch, choose 'Done' without adding any extras, and most of the remaining steps will be omitted.

Some Desktop Environments include their own Display Manager (greeter and login screen), and for those that don't, Feliz will offer a few options. Again, this is optional.

## **Hostname**

In the next step, a hostname is needed by which your computer will be recognised on a network. This can be almost anything you like, but is probably best kept simple. If you just hit [Enter] without typing anything, the installer will allocate the hostname 'arch-linux'.

## **Username**

We are nearly done. If you have chosen to install a DE or other extras, the installer will now ask you to enter a name for the main user of the system. Keep it to one word, and all lower-case. If you don't enter one, the installer will create one called 'archie'.

## **Virtualbox**

If you are installing your new Arch system in Virtualbox, Feliz will offer to install the Virtualbox guest utilities. These are worth having, as they help with screen settings and so on.

## **Partitioning**

Feliz looks at your hardware, and reports any partition table it finds. If there are no partitions on the device, the installer will offer to automatically create some for you, or you can use cfdisk, a basic but useful partitioning tool, to do it yourself. If there are existing partitions, you can choose to use some or all of them, either by selecting cfdisk to modify them yourself, or by leaving them as they are and proceeding to the next step of allocating them. If you choose to allow the installer to partition the device, you should be aware that it will use the whole disk, destroying any existing data on it.

The minimum number of partitions on traditional MBR (BIOS) hardware is one, the root or / partition, onto which the system will be installed, and which will also hold the user's home folder, if no separate /home partition is created. On EFI systems, a separate EFI partition and a

/boot partition are also required. Some kind of swap facility is also advised, but is not essential, especially if you have plenty of ram and don't run memory-intensive applications. If you are tight for disk space, you can choose to allocate a swap file instead of making a swap partition, and the installer allows you to do that and to set its size.

After you have defined your partitions, the installer helps you to allocate them as required by your hardware.. If you asked the installer to partition the device for you, this step will be bypassed.

### **Kernel**

Partitioning taken care of, the installer moves on ask which kernel you prefer to use. In my opinion, there are few advantages in going for the latest kernel, unless you really need the latest drivers for your devices, and has sometimes caused me problems when running an installation in Virtualbox. Generally, LTS (Long Term Support) is fine, and perhaps more stable, but the choice is yours.

### **Mirrors**

Here you can select the location of the best mirrors from which to download files.

### **Confirmation**

And that's it! The next screen will list all the settings you have entered. If you are happy with everything, just press [Enter] to go ahead and install with those settings. Or, if there's anything you want to change, enter the number of the item in the menu to select it and run that stage again.

### **Installation**

From this moment on, Feliz sets to work using the information you have provided to automatically install your new system without any further intervention. Feliz tells you what is happening at each step, so you can follow it on the Arch wiki Beginners' Guide if you like.

At the end, Feliz tells you how long it has taken (this will vary depending on the time you spend looking around the categories, your internet connection speed and the number of items on your 'shopping list').

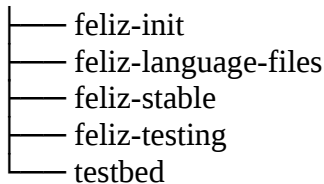
### **Completion**

All that remains, after installation has finished, is to enter a password for root (the system administrator) and the user, then restart into your new system – remove the virtual CD, or choose “Boot existing OS” from the boot menu.

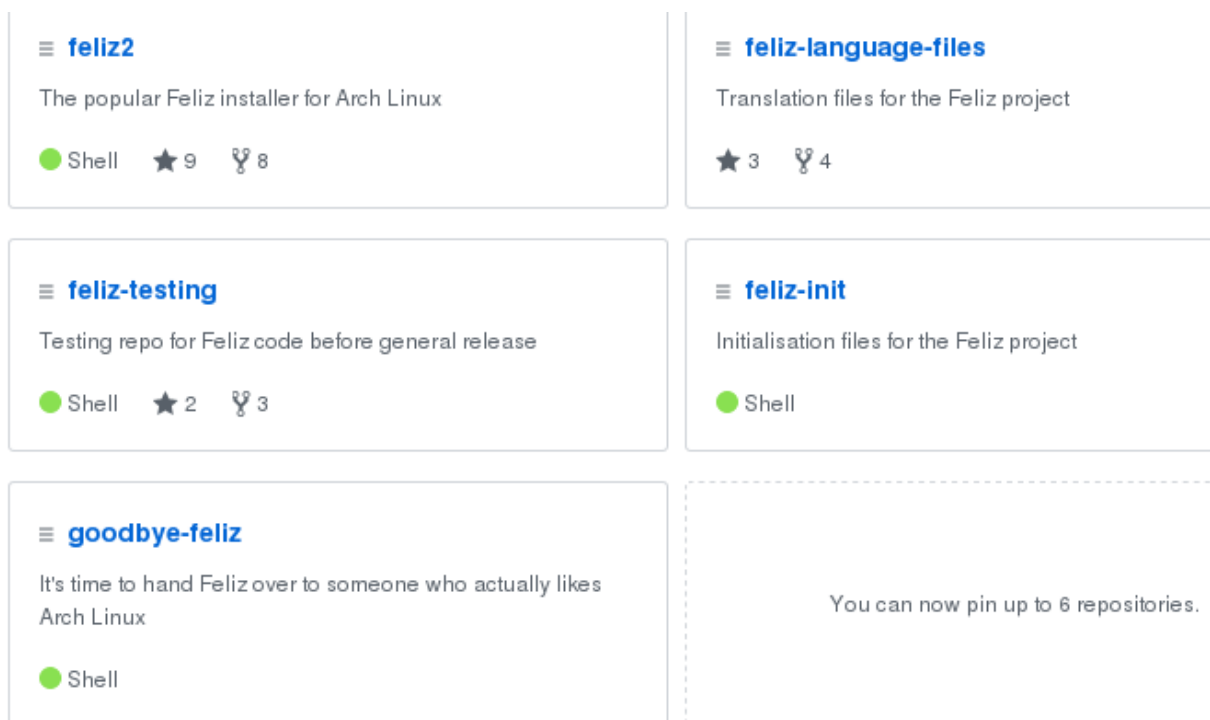
## 4 – Developing Feliz - Directory Structure

All Feliz development occurs on scripts in one of five ‘workshops’ on my computer. Most of these are tracked by the Git version control system (more on that later) and uploaded to my repositories on Github.com for sharing with, well, anyone who wants to try them. When the code has been tested to my satisfaction in the ‘testing’ repo, it is copied into another, exclusively for the stable branch.

Here’s a snapshot of all the Feliz workshops ...



There are five working directories (think of them as virtual laboratories): one for the initialising scripts, one for the translation files (more on those later), one each for the testing and stable branches, and one (testbed) where I test new code locally. The first four are maintained with Git, corresponding to the four repositories on Github – **feliz-init**, **feliz-language-files**, **feliz2** (the stable branch) and **feliz-testing** ....



They have been joined on Github by a repository created for this handover

The felizinit modules are the only scripts that are compiled into the distribution iso. They serve simply as a 'launcher', to test the internet connection and download the latest version of the intallation scripts. When a new iso is being prepared, either the felizinit-stable or felizinit-testing script is renamed just 'felizinit' and packaged into the iso to be launched when the Arch session has been started.

Within each of the two working laboratories (feliz-testing and feliz-stable) there is a number of configuration files that are copied into the user's system during installation, and the scripts themselves, which are split into manageable-sized files ...

- autostart
- cities.list
- compton.conf
- config
- conkyrc
- desktop-items
- face.png
- feliz
- feliz.png
- feliz.sh
- f-part1.sh
- f-part2.sh
- f-run.sh
- f-set.sh
- f-vars.sh
- keymaps.list
- languages.list
- libfm.conf
- LICENSE
- listgen.manual
- listgen.sh
- lxdm.conf
- menu.xml
- panel
- rc.xml
- README
- themes
  - egtk
    - openbox-3
      - max\_toggled.xbm
      - max.xbm
      - themerc
      - themerc~
- wallpaper.jpg

Each laboratory contains copies of the same files, at different stages of development, with the exception of the feliz-testing directory, which has a file called TESTING instead of README.

Both **feliz-stable** and **feliz-testing** use the same **feliz-language-files**.

As we progress through this book, I will explain the reasons for that layout and the purpose of each 'laboratory', and I will explain how some important parts of the code work. However, at this point I want to mention what I regard as 'good practice' when coding:.

- 1) All error output during installation is redirected to a log file 'feliz.log' to allow analysis;
- 2) Wherever necessary, operations that are performed frequently are housed in functions and called as required. This makes for easier code understanding and maintenance;
- 3) Functions are 'stepped' (ie: indented progressively) to help with code maintenance;
- 4) Code is commented as much as possible, for the same reason;
- 5) Each script has the extension .sh, to identify it as a script. There is also one called just 'feliz' which is an identical copy of 'feliz.sh' and exists only for compatibility with older versions of the iso.

The six shell scripts are:

- feliz.sh – the main script, which calls the functions contained in the others;
- f-part1.sh – containing functions for MBR partitioning;
- f-part2.sh – similar, but for GPT partitioning;
- f-run.sh – functions used during the installation phase;
- f-set.sh – functions used during the data-entry phase;
- f-vars.sh – where all the variables and arrays are declared;

In addition, there is the listgen.sh script, which contains the functions I created to enable Feliz to display the various kinds of lists during the data-entry stage, and 'listgen-manual', a text file containing advice for anyone wishing to incorporate the listgen functions into their own scripts.

All the other files in each directory are either used by the Feliz functions for lookup, or are copied by Feliz into a new installation (some of them specific to the configuration of the FelizOB desktop). These will be explained later.

There are also a copy of the GNU General License, under which Feliz is distributed, and a README.

In the remaining chapters of this book I will explain how the Feliz scripts work, and the procedure I follow to generate the Feliz iso.

## 5 - The Feliz Initialising Script

At the top of the chart in chapter 5, showing the layout of my 'workshop', you will see the **feliz-init** laboratory, containing two small scripts that launch Feliz when the iso is run. They consist of code that checks the internet connection, downloads the latest scripts from Github and then starts **feliz.sh**.

These two scripts (**felizinit-testing** and **felizinit-stable**) differ only in the sets of files they download from Github - either from the testing branch or the stable branch.

Roughly once a month, I generate a new release of the Feliz iso which contains the **felizinit-stable** script in a copy of the latest Arch iso. This changes very little, and you don't have to get the latest iso to use Feliz; any copy of the iso will download and run the latest stable installation scripts from Github, saving you the need to get a new iso every time something changes in the installer.

If you care to take a trip to the feliz-init repository at Github, you can examine the code:

<https://github.com/angeltoast/feliz-init>

Github is a great place to read code and learn from those who have already created something.

## 6 – Feliz Development - File Structure

In broad outline, the main Feliz script (`feliz.sh`) calls the functions in the other modules as required. It operates in two distinct phases. First it gathers all the necessary data. Most of the code for this phase of the installation is contained in the script `f-set.sh` (short for ‘feliz-settings’), and the global variables and functions in `f-vars.sh`. In this stage, almost all the data collected is stored in variables for later use.

Then, after the user has selected their preferences from all the options, `feliz.sh` proceeds to the second phase, in which it automatically performs the installation, using the variables set in the first phase, requiring no further intervention. It is this aspect of the Feliz scripts that makes it so fast, as the user does not have to wait while each step is performed. You can go and make a cup of tea or mow the lawn, leaving Feliz to do the job.

There is no point in me examining and dissecting each of the scripts here, it would take hundreds of pages and would be boring to read; it would also be out of date, because Feliz is constantly advancing. The best thing you can do is to clone the **feliz2** repository from Github and read the scripts. If you have Git installed, you can ...

```
git clone https://github.com/angeltoast/feliz2.git feliz
```

If not, follow these steps ...

- 1) Download the complete stable archive from Github:

```
wget https://github.com/angeltoast/feliz2/archive/master.zip
```

- 2) And unzip the archive:

```
unzip -o master.zip
```

However, an explanation of the file structure I use for Feliz may be helpful.

When **felizinit** has finished preparing the environment, it starts **feliz.sh**, the main script, which controls all the rest of the processing. One of the first things **feliz.sh** does is to source the other scripts, so that the functions in them are available ...

```
# Include source files
```

```
source f-vars.sh    # Most variables and arrays are declared here
source f-set.sh     # Functions to set variables used during installation
source listgen.sh   # Menuing functions
source f-part1.sh   # Functions concerned with allocating partitions
source f-part2.sh   # Guided partitioning for BIOS & EFI systems
source f-run.sh     # Functions called during installation
```

Essentially, functions that tend to work together are grouped together. So, for example, the functions concerned with the user's settings and choice of extra applications are held in the **f-set.sh** module, while the complex functions that help the user with partitioning are (you guessed it) in the **f-part1.sh** and **f-part2.sh** modules.

The exception is the **f-vars.sh** script which, as its name suggests, takes care of defining and initialising the main variables and arrays used throughout Feliz, plus a few functions that are shared by the other units - mostly handling input/output. Strictly speaking, shell scripts in Bash don't have to declare variables before use - Bash is quite happy for them to be created as

required - but I find it helpful to initialise (set default values) to prevent any holes in the data later.

The seventh script is worth special mention, however.

In the early days of developing Achoo, I used the Bash 'select' function to generate numbered lists for users to choose which option they wanted. The problem with 'select' is that it aligns everything against the left edge of the screen, which does not make for a balanced display. So I wrote the **listgen** functions to serve the same purpose but in a more attractive way.



## 7 – Developing Feliz - Listgen

The **listgen** functions have been an integral part of the Feliz installer since its birth, and have grown with it. They are designed to help with creating intuitive cli interfaces without using Dialog, Whiptail, or any other external package. Instead, the tput library is used to determine the terminal dimensions and to display lists and text centrally according to content.

There are three main functions: **listgen1** is for situations where a simple one-word menu item is sufficient, while **listgen2** is for use where more descriptive text is necessary. Both functions centre the text on the screen in an aligned, interactive menu. In addition, **listgenx** exists for those situations where there would be too many options to display vertically. It takes a long list of one-word options, and assembles them into as many columns as will fit in the terminal. If there are more items than will fit a single screenful, the user can navigate to the next page until they find the one they want. Each item is numbered, and the user enters the number of the item they wish to select. The **listgen** functions are designed to be application-independent, so the listgen.sh module can be included in any set of scripts, or the functions can be copied into your script.

The full list of functions in listgen is:

- Heading – Prints the contents of a variable \$Backtitle (which may be set elsewhere in your scripts) in a banner at the centre-top of the terminal or screen.
- first\_item – Used by the list-printing functions to centralise the first item of a menu;
- subsequent\_item – Used by the list-printing functions to align successive menu items;
- PrintRev – Reverses (ie: highlights) text colour for selected menu item and/or button;
- Buttons – Prints a row of buttons;
- ActiveMenu – Controls the selection and highlighting of menu items;
- listgen1 – Generates a menu of one-word items;
- listgen2 – Generates a menu of multi-word items;
- listgenx – Generates pages of numbered lists, for lists that would exceed screen size;
- SelectPage – Selects appropriate page according to user input;
- PrintPage – Displays the selected page of data, fitting as many columns as possible according to the terminal/screen size.

### A simple example, using the 'Buttons' function

Buttons can be used to generate an interactive Yes/No display.

A call to buttons uses three arguments:

- 1) Type (Menu, Yes/No);
- 2) Button string. This should contain one or two words: eg: 'Ok' or 'Ok Exit'
- 3) Message string.

At the time of writing, Buttons is limited to displaying only two buttons, but this may be extended to three options in the near future.

Call Buttons with a line like this:

*Buttons "Yes/No" "Yes No" "Use arrow keys to move. [Enter] to select"*

The arguments in the example above inform the Buttons function of the type of operation, that the the buttons are to be labelled 'Yes' and 'No', and a short line of guidance to the user is to be displayed.



Example of the use of the buttons function

## The listgen1 Function

The first and simplest listing function, **listgen1**, is for use in shell scripts to generate an interactive menu. It prints the list of options provided, in the form of a scrollable menu, with a pair of 'buttons' to confirm or exit.

Arguments:

- 1) A simple string array variable of the items to be listed;
- 2) An optional one-line string of text to be displayed beneath the 'buttons', advising the user how to use the menu. If this argument is passed as an empty string, the menu will display without the instructions line. This is for situations where a menu may be close to filling the screen, or where no instructions are necessary;
- 3) Button text eg: 'Ok Exit' or just 'Ok'. These must be single-word options. If only one word is passed (eg: 'Ok') then no exit option will be printed; in this way you can make sure that your users pick one of the menu items offered.



Example of listgen1

A call to listgen1 takes the form ...

```
listgen1 "argument1" "argument2" "argument3"
```

All three arguments must be passed, although the second argument can be an empty string, if no message is required.

Example call: `listgen1 "Reboot Shutdown" "" "Ok Exit"`

The first (string array) argument may be passed in one of three ways:

- 1) It may be specified in the calling line (as above): `listgen1 "item1 item2 item3 ... "`
- 2) It may be assigned to a variable: `VariableName="item1 item2 item3 ... "`

For example ...

```
Accessories="Conky Geany Nautilus Terminator"
listgen1 "$Accessories"
```

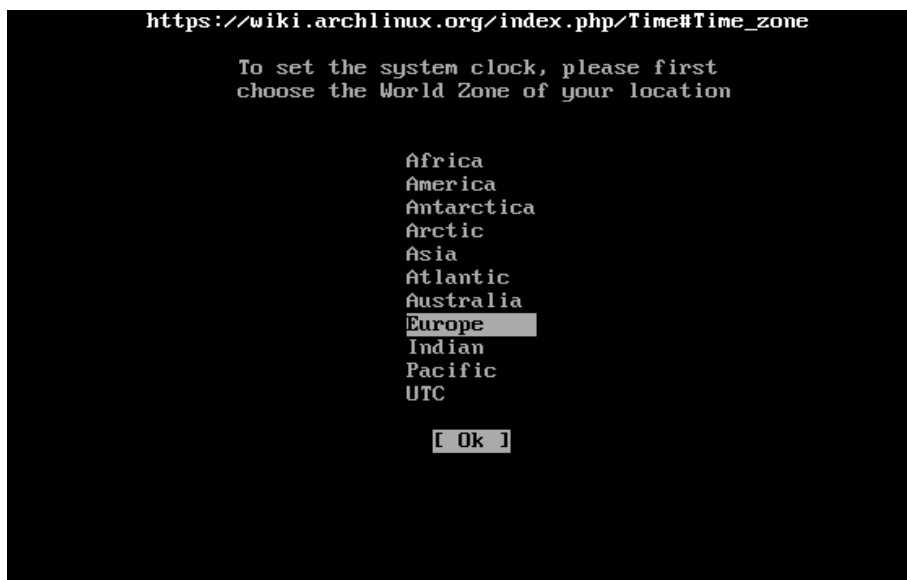
3) It may be generated by a bash command ...

For example ...

```
Partitionlist=$(lsblk -l | grep 'part')  
listgen1 "$Partitionlist" "" "nocancel"
```

Note: A bash-generated list should be converted to a string array. For example:

```
Zones=(`timedatectl list-timezones | sed 's/\./.*$/ ' | uniq`)  
passzones=""  
for z in ${Zones[@]}; do  
    passzones="$passzones $z"  
done  
...  
listgen1 "${passzones}" "" "nocancel"
```



How the 'zones' example might look

### Variables set by listgen

All the listgen functions set two global variables for use by the calling function:

- 1) \$Response - the item number selected by the user, and
- 2) \$Result - the label (from the variable array) of the item selected.

This means that you can respond to the user input by using (for example) a 'case - esac' statement or an 'if - else- fi' statement.

For more complex lists such as are needed for the user to choose a desktop and other extra applications, I wrote listgen2.

## The listgen2 Function


listgen2 is designed for displaying longer descriptions in a menu. It receives the text of the long items via an array. When an item is selected, listgen2 finds that item in a reference array of short (one-word) names that match the array of long-descriptions, and saves it as the global variable \$Result which may then be used by the calling function if desired. Although complex to set up, it adds greater functionality and user-friendliness for menus. The reference array must be specifically declared, and the elements filled, before it can be used.

Calling listgen2 is similar to calling listgen1, but with an added 4<sup>th</sup> argument:

```
listgen2 "$PrimaryFile" "argument2" "argument3" "ReferenceArray"
```

- 1) The primary file. That is, a string of single-word references, as with listgen1;
- 2) A short message to be displayed to help the user. This may be left empty;
- 3) The button text eg: 'Ok Exit' or just 'Ok';
- 4) The name (only) of the reference array (this will be used to access the array).

Four arguments **MUST** be passed, but only 1) and 4) are essential – 2) and 3) are optional and may be blank strings (but **MUST** be included).



Example of listgen2

Set up the primary file like this:

```
PrimaryFile="item1 item2 item3 item4"
```

Set up the array like this:

```
Array[1]=" This is item one"  
Array[2]="This is item two"  
Array[3]="This is item three"  
Array[4]="This is item four"
```

Listgen2 returns the same global variables as listgen1 and uses the same built-in functions

## The listgenx Function

Listgenx differs from the other two in that, instead of a cursor-driven menu, it generates a numbered list from which the user can choose by entering the number displayed next to the item. This is provided for situations where a large number of options is to be offered, and a listgen1 or listgen2 menu would scroll off the screen.

Because listgenx must redraw the screen for each pageful of data to be displayed, it must be provided with all the text it needs for user advice and prompting. As a result, listgenx requires five arguments

Before calling listgenx, the calling function must generate a file containing the items to be displayed. The items in the file must be only one word each, must be one to a line, and the file must be named **input.file**

The input.file may be generated in any way that produces a list of items, one on each line:

1) It may be passed directly:

```
printf "%-s\n" "item1" "item2" "item3" > input.file
```

2) It may be generated in a loop:

```
for i in names.list
do
    echo $i >> input.file
done
```

3) It may be generated by a bash command ...

```
lsblk -l | grep 'part' > input.file
```

The calling function can send optional 'Headline' string as the first argument, to appear at the top of the screen, and an optional 'Message' to appear at the bottom of the screen.

**listgenx** is called in the form:

```
listgenx "argument1" "argument2" "argument3" "argument4" "argument5"
```

Argument1 is an optional string of text to be displayed at the top of the screen as information for the user. If passed as a null string, no message will be printed;

Argument2 is an optional instruction line to appear at the prompt, which will be printed at the bottom of the terminal/screen, beneath the pageful of data – for example:

```
"Please enter the number of your selection: "
```

If argument1 is empty, no instruction will be printed – this could be confusing for the user, so it is advised that a message similar to that above is sent;

The remaining three arguments are paging advice ...

Argument3 allows for the user to back out of listgenx and return to the calling function without making a selection – for example:

```
"or ' ' to go back"
```

Argument4 and argument5 inform the user that they can page forwards or back through the data by entering either '>' for the next page, or '<' for the previous page. listgenx will

intelligently display these messages only if appropriate for the circumstances (ie: it will not suggest '>' if there are no more pages to be displayed. The arguments may be as follows:

"Enter '<' for previous page"

"Enter '>' for next page"

**listgenx** will display as many columns as can be fitted onto the width of the terminal, instructing the user to enter a number for the item selected, or < or > to display the previous or next page of data, if relevant.

**listgenx** sets the same output variables as **listgen1** and **listgen2**

```
https://wiki.archlinux.org/index.php/Mirrors

Please choose a country
1) Australia      30) Kazakhstan    58) Vietnam
2) Austria        31) Latvia
3) Belarus        32) Lithuania
4) Belgium        33) Luxembourg
5) BosniaHerzegov 34) Macedonia
6) Brazil         35) Netherlands
7) Bulgaria       36) New_Caledonia
8) Canada         37) Norway
9) Chile          38) Poland
10) China         39) Portugal
11) Colombia      40) Qatar
12) Croatia       41) Romania
13) Czech_Republic 42) Russia
14) Denmark       43) Serbia
15) Ecuador       44) Singapore
16) Finland       45) Slovakia
17) France        46) Slovenia
18) Germany       47) South_Africa
19) Greece        48) South_Korea
20) Hong_Kong     49) Spain
21) Hungary       50) Sweden
22) Iceland       51) Switzerland
23) India         52) Taiwan
24) Indonesia     53) Thailand
25) Iran          54) Turkey
26) Ireland       55) Ukraine
27) Israel        56) United_Kingdom
28) Italy         57) United_States
29) Japan         58) Vietnam

Please enter the number of your selection:
or ' ' to exit
```

### Listgenx

Feliz is in constant development, and has recently become multi-lingual, but great care has been taken to keep Listgen separate from the translation code. This is so that you can use Listgen in any situation. Nevertheless, in Listgen, as in Feliz, prompts and button-text were all in English, and an extensive rewrite of the Listgen functions has been necessary to allow any calling functions to specify exactly the text to be displayed in and below the lists.

## 8 - How Feliz Became Multilingual

You know how it is, something that seemed like a good idea at the time becomes a nightmare. That is what happened to me the first time I tried to make Feliz accessible to a wider audience, by making it multi-lingual.

I looked at how others have done it, and every example I read used the following method:

- 1) A set of language files is created, one for each translation;
- 2) Each of those is a script, containing variable declarations, one for each line of text that is to be displayed when the main script is run;
- 3) The main script (or scripts) is written (or adapted) so that, when text is to be displayed, it references the variable set for that text.

Despite some trepidation, I set about following their lead, adapting Feliz to use the same system.

Let's just say things got complicated. Maybe I don't have the right kind of brain. But, whatever the reason, as it progressed, I found that I couldn't even follow my own scripts. I very soon ditched the idea and reverted my code to the last saved version.

Then, late in 2016, a member of one of the Feliz online communities asked if Feliz could be translated, and I decided to have another look at how it might be done. After a few days, I came up with the ideas incorporated now into Feliz and described in this little book.

I should, however, first point out that this method is (probably) only suitable for 'European' style languages – ie: those that are written in lines from left to right. I cannot see how it could work with Arabic or Oriental writings. With that proviso, here is my method.

### A Summary of the Feliz Method

- 1) Like the other method, described above, a file is created for each language;
- 2) However, in the Feliz method, these files are plain text files, not scripts;
- 3) In every file, each line corresponds. In other words, line number 22 in the Portuguese language file translates line 22 in the English language file, and the Polish language file.
- 4) Feliz contains two simple functions:
  - (a) SetLanguage; and
  - (b) Translate

The '*SetLanguage*' function is called only once, at the start of the script. It defines the 'base' language to be used in the script, and offers the user of the script the opportunity to choose which language they wish the script to display while running. The 'base' language is simply the language of choice for the programmer – in the case of Feliz, this is English, so English.lan is the 'base' language file.

The '*Translate*' function does just what its name suggests – at any point in the script where text is to be displayed, Translate is called to find the appropriate line in the language file chosen by the user, and to display that text.



### **First Choose the 'base' Language**

For me, this had to be English, as it is the only language in which I am fluent. It also happens to be the language in which Arch Linux is fluent.

I used some Bash stuff to extract every line of text that is displayed by the seven scripts that make up Feliz, sorted them, removed duplicates, and put them into a file named English.lan (the extension is necessary – it can be whatever you like, but must be consistent, as the language files will be listed by 'SetLanguage').

Then I asked in the Feliz Facebook group for volunteers, and several members kindly agreed to translate English.lan into their language. I asked them to name the files in their language, not in English, so the Polish file is Polski.lan, Portuguese is Português.lan, the Spanish file is Español.lan, etc.

Now, the beauty of this method is that any one of the language files can be the 'base' language. If, for example, someone wishes to clone Feliz to make their own installer, they can (if they wish) use all the existing language files, but can choose to program in (for example) Polish, and use the 'Polski.lan' file as the 'base' file.

## The 'SetLanguage' Function

```
SetLanguage() {
    _Backtitle="Feliz - Arch Linux installation script"
    print_heading
    PrintOne "" "Idioma/Język/Language/Langue/Limba/Língua/Sprache"
    Echo
    listgen1 "English Deutsche .... Italiano Nederlands Polski" "" "Ok"
    case $Response in
        2) InstallLanguage="de"
            LanguageFile="German.lan"
            ;;
        3) InstallLanguage="el"
            LanguageFile="Greek.lan"
            setfont LatGrkCyr-8x16 -m 8859-2
            ;;
        4) InstallLanguage="es"
            LanguageFile="Spanish.lan"
            ;;
        5) InstallLanguage="fr"
            LanguageFile="French.lan"
            ;;
        6) InstallLanguage="hi"
            LanguageFile="Hindi.lan"
            setfont viscii10-8x16 -m 8859-2
            ;;
        7) InstallLanguage="it"
            LanguageFile="Italian.lan"
            ;;
        8) InstallLanguage="nl"
            LanguageFile="Dutch.lan"
            ;;
        9) InstallLanguage="pl"
            LanguageFile="Polish.lan"
            ;;
        10) InstallLanguage="pt-PT"
            LanguageFile="Portuguese-PT.lan"
            ;;
        11) InstallLanguage="pt-BR"
            LanguageFile="Portuguese-BR.lan"
            ;;
        12) InstallLanguage="vi"
            LanguageFile="Vietnamese.lan"
            setfont viscii10-8x16 -m 8859-2
            ;;
        *) InstallLanguage="en"
            LanguageFile="English.lan"
    esac
    # Get the required language files
    wget https://raw.githubusercontent.com/angeltoast/feliz-language- files/master/English.lan 2>>
        feliz.log
    if [ $LanguageFile != "English.lan" ]; then # Only if not English
        wget https://raw.githubusercontent.com/angeltoast/feliz-language-files/master/$
            {LanguageFile} 2>> feliz.log
    fi
    Common # Call function to set some common translations
}
```

Notes:

- 1) *print\_heading* is a small function that displays a message, centred at the top of the screen, saying:

Feliz - Arch Linux installation script

When a language has been chosen, this text will be translated into the user's selected language;

- 2) The *PrintOne* function, called after *print\_heading*, is another Feliz function, which prints its output centred on the screen or terminal. This function can also be found in *f-vars.sh*, along with its companion *PrintMany*, which starts its output line level with the *PrintOne* displayed above it, to create a neatly stacked list;
- 3) After an echo, *listgen1* is called (see the chapter on *Listgen*)
- 3) When the user has made a selection from those offered through *listgen1*, *SetLanguage* takes the value returned by *listgen1* via the variable *\$Result* and assigns the selected language (or English, if no language was chosen);
- 4) The *\*)* part of the case structure sets the *LanguageFile* variable, which will then be used throughout Feliz by the *Translate* function;
- 5) The last line calls a function that uses *Translate* to prepare some variables containing a few of the most commonly used words and phrases in Feliz; this is, of course, personal and optional.

### The three functions mentioned above

```
print_heading() { # Always use this function to clear the screen
  clear
  T_COLS=$(tput cols) # Get width of terminal
  LenBT=${#_Backtitle} # Length of backtitle
  HalfBT=$((LenBT/2))
  tput cup 0 $((T_COLS/2)-HalfBT) # Move cursor to left of center
  tput bold
  printf "%-s\n" "$_Backtitle" # Display backtitle
  tput sgr0
  # printf "%$(tput cols)s\n"|tr ' ' '-' # Draw a line across terminal
  cursor_row=3 # Save cursor row after heading
}
```

```
PrintOne() { # Receives up to 2 arguments. Translates and prints
             # text centred according to content and screen size
  if [ ! "$2" ]; then # If $2 is missing or empty, translate $1
    Translate "$1"
    Text="$Result"
  elif [ $Translate = "N" ]; then # Don't translate any
    Text="$1 $2 $3"
  else # If $2 contains text, don't translate any
    Text="$1 $2 $3"
  fi
  local width=$(tput cols)
  EMPTY=" "
  stpt=0
  local lov=${#Text}
  if [ ${lov} -lt ${width} ]; then
    stpt=$(( (width - lov) / 2 ))
    EMPTY="$(printf '%*s' $stpt)"
  fi
  Echo "$EMPTY $Text"
}
```

```
PrintMany() { # Receives up to 2 arguments. Translates and prints text
              # aligned to first row according to content and screen size
  if [ ! "$2" ]; then # If $2 is missing
    Translate "$1"
    Text="$Result"
  elif [ $Translate = "N" ]; then # Don't translate any
    Text="$1 $2 $3"
  else # If $2 contains text, don't translate $1 or $2
    Text="$1 $2"
  fi
  Echo "$EMPTY $Text"
}
```

### The 'Translate' function

This little function is the heart of the translation system. It is called every time text is printed on the screen ...

```
Translate() { # Called by PrintOne & PrintMany and by other functions as required
              # $1 is text to be translated
  Text="${1%% }" # Remove any trailing spaces
  if [ $LanguageFile = "English.lan" ] || [ $Translate = "N" ]; then
    Result="$Text"
    return
  fi
  # Get line number of "$Text" in English.lan
  #          exact match only | restrict to first find | display only number
  RecordNumber=$(grep -n "^${Text}$" English.lan | head -n 1 | cut -d':' -f1)
  case $RecordNumber in
    "" | 0) # No match found in English.lan, so use the passed text
      Result="$Text"
      ;;
    *) # Read item from target language file
      Result="$(head -n ${RecordNumber} ${LanguageFile} | tail -n 1)"
  esac
}
```

## 9 - Thank You Linus Torvalds For Git

Strictly speaking, as an independent developer, working alone, I don't really need a version-control system. However, Git has proved to be invaluable in helping me to keep track of changes, and makes it simple for me to share my code with anyone who wants it, through Github.com

Git was developed by Linux Torvalds, the creator and project leader of the Linux kernel, when he became frustrated with the other, commercial, version control systems he was using. Like the kernel, Linus has made Git freely available.

Although there are some GUI front-ends for Git, it is really at its easiest and most productive when used from the command line. My way of using it may not fit exactly into the recommended procedures, but it works for me. Obviously, if you are planning to use it, Git must be installed on your computer - it is in the repos of just about every distribution, so this should be easy.

To start a new project, I first log in to my account at Github and create the empty repository there. Then, in the directory in my workshop where the code for the project is held (or is to be built) I follow the following simple steps:

- 1) Initialise the project in that directory:

```
git init
```

- 2) Set up your user details:

```
git config --global user.name YourGitUserName
```

```
git config --global user.email your@email.address
```

You may also want to set your preferred text editor:

```
git config --global core.editor YourEditor
```

- 3) To get started, I like to first add a plain-text README file:

```
touch README
```

- 4) After typing some stuff into that file, it can be added, along with any existing files in that directory, to a 'snapshot' of the file structure in Git:

```
git add .
```

Note the dot (.) on that line. It tells Git to 'add everything here'. This adds new files to the snapshot. The snapshot is now stored in a temporary staging area which Git calls the "index"

- 5) You can permanently store the contents of the index in the repository with git commit:

```
git commit -m "first commit"
```

The -m option is to tell Git to include the message in quotes ("first commit" in this case) with the commit. At this stage, we are simply committing the files; the repos at Github have not yet been updated. If there were no new files to be added (in other words, you are just updating existing files) you can just use *git commit -a* to select all files to the commit.

After entering this command, Git will open your default text editor for you to select which files you wish to update. Simply delete the hash symbol (#) before each line to be committed, then save.

- 6) Before the committed files can be pushed up to Github, it is necessary to establish your Github credentials on the local computer. I do this using HTTPS by entering:

```
git remote add <tag> https://github.com/<user>/<project>.git
```

In this context, 'tag' is a one-word identifier. If you don't enter one, Git assigns 'origin' (I prefer to use a name that clearly identifies the part of the project I am maintaining – eg: 'stable' or 'testing'). The remainder of the path is your Github user name and the project name at Github. For example, when registering the Feliz testing branch with my Git credentials, I would have entered:

```
git remote add testing https://github.com/angeltoast/feliz-testing.git
```

... and would be prompted for my Github password. Once that has been accepted, Git has all the information it needs to maintain the link between my local files and those stored in my repository on Github.

- 7) The final stage to uploading the committed files to Github is:

```
git push <origin-or-whatever> master
```

Once the preliminaries above are set, subsequent updates are easy:

|                               |                                 |
|-------------------------------|---------------------------------|
| <i>git status</i>             | To check the current situation; |
| <i>git add .</i>              | Add any new files;              |
| <i>git commit -a</i>          | Edit the commit message;        |
| <i>git push origin master</i> | Push to Github.                 |

There is, of course, much more to Git than these simple, basic steps, but these are all I use as a simple country girl working on my code alone. The 'man' pages are there to guide you. Start with 'man git', or 'man git tutorial', or 'man git everyday' to start to learn more.

## 10 – Using and Building the Feliz ISO

Users wishing to access the Feliz scripts can do it in one of two ways. You can use them with an Arch Linux iso, as follows:

- 1) Download the latest Arch Linux iso from ... <https://www.archlinux.org/download/>
- 2) Then either:
  - (a) Burn the Arch iso image to flash media as described here ... [https://wiki.archlinux.org/index.php/USB\\_flash\\_installation\\_media](https://wiki.archlinux.org/index.php/USB_flash_installation_media)  
... or...
  - (b) To test in Virtualbox or other virtualising system, add the iso as a disk.
- 3) Start an Arch session, then when you have the root prompt, install unzip by entering the following:  

```
pacman -Sy unzip
```

  
(unzip is needed by felizinit to inpack the archive)
- 4) Next enter:  

```
wget https://raw.githubusercontent.com/angeltoast/feliz-init/master/felizinit-stable
```
- 5) Now enter:  

```
chmod +x felizinit-stable
```
- 6) Then start Feliz by entering:  

```
./felizinit-stable
```

The second way of running Feliz is to download a Feliz iso which contains both Arch Linux and the felizinit script. The script can be obtained either from Sourceforge.net ...

<https://sourceforge.net/projects/feliz>

... or Github.com ...

<https://github.com/angeltoast/feliz/releases>

~ ( # ) ~

The remainder of this chapter is devoted to explaining how I create that Feliz iso. I wish to make it clear that this is just *my* method, the way I have found that works for *me*. I am not saying that this is the right way, nor is it the only way, nor the ‘best’ way (and maybe it is not even a good way, but it works).

If your main system is not Arch, then you will need to create a virtual environment in which to work. I use Virtualbox, and I have made a 30GiB virtual drive for my Feliz iso work; if you follow this route, don’t make the virtual drive too small – 30GiB has proved to be ok for me, but make yours bigger if you wish. On the virtual machine, install Arch Linux (use Feliz – why not, it’s fast and easy), then in that environment ...

First, it is necessary to install a bunch of scripts from Arch Linux called ‘Archiso’ - these are the same scripts that the Arch devs use to build their own iso each month. They are in the Arch repos, so it is simply a matter of entering (as root):

```
pacman -S archiso
```

This creates a directory /usr/share/archiso/ and places various files in it.

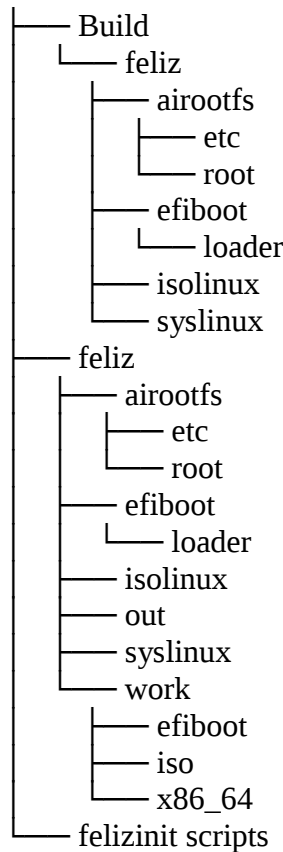
To properly understand what follows, I strongly recommend that you read the Arch Wiki on Archiso ... <https://wiki.archlinux.org/index.php/Archiso>



Next, create a special directory in your home directory in which to hold the components for generating the iso. I call mine 'Build'

The 'Build' directory is not to be used to create an iso, just to prepare the 'feliz' folder. You will place all the ancillary files in Build, but the actual iso creation is to take place in a ~/feliz folder created specially for that purpose.

Here is an illustration of how the directory tree structure looks in my Feliz ISO Workshop ...



You can see that the 'feliz' directory in my home directory is a mirror of the one in 'Build' with the exception that (after generating an iso) it contains a 'work' folder and an 'out' folder, in which Archiso puts the completed iso.

That 'feliz' directory is a copy of the 'releng' directory created during the installation of Archiso, adapted to suit Feliz. The changes are detailed in a text file named 'Customising Archiso' included on the Github page with this pdf. There is also a script I wrote which automates the process, called 'update-archiso'. That script (which lives in the Build directory) makes all the desired changes to all the scripts and files for Feliz each time Arch Linux release an update to Archiso. It only has to be run if 'archiso' has been updated, but MUST be run as root. A copy of the 'update-archiso' script is included on the Github page with this pdf.

The 'update-archiso' script copies the releng folder into 'Build', renames it 'feliz', changes certain lines of files in the syslinux folder to read 'Feliz' instead of 'Arch Linux', copies over the Feliz splash screen, appends the felizinit script name to archiso\_sys64.cfg and adds a 'skel' directory to hold felizinit. It also adds a couple of programs needed by the installer to the packages.both file.

That's all the preparation. To create an iso, the procedure is as follows:

***Start in the Build directory ...***

1) If Archiso has been updated by Arch Linux, run ‘update-archiso’;

(Note that ‘update-archiso’ makes a safe copy of any existing ‘feliz’ folder in Build)

***Next move to the /home/<user> directory ...***

2) Delete any existing 'feliz' directory;

3) Copy the 'feliz' folder from the Build directory;

4) Copy felizinit or felizinit-testing into ~/feliz/airootfs/etc/skel/ as felizinit;

5) Close the file manager if in use;

6) Open a terminal, su to root, go to ~/feliz and run ./build.sh;

7) When the process is successfully finished, revert to user and copy the output file from ‘out’ into a suitable folder for uploading to Sourceforge, Github, or wherever.

Note: The build process takes a long time (from 15 minutes to an hour, depending on your processor. Do not worry – if anything goes wrong, it will show an error message.

**return 0; done; }**

That’s it from me.

Thank you to everyone who has helped to make Feliz what it is.

Goodbye.

Liz