THE UNIX PHILOSOPHY

(1)
This talk is about The Unix Philosophy.

(2)
There is no Unix philosophy; or rather the Unix Philosophy is not a product of philosophical exposition – it was created bottom-up from experiences, sharing knowledge, a sort of hacker folklore. The Unix philosophy was not created top-down, exposition came later.

What is The Unix Philosophy? It is a set of beliefs, values, and heuristics on how a computer should be used by a qualified user. Unix is not an application, it is a collection of applications - tools - means for putting tools into productive *relationships*, compositions.

(3)
Few or none Unix systems are running today. But Unix has survived through Linux, and indirectly more or less power the web, and almost all embedded and IoT devices (as well as random Desktop environments such as this one). MacOS is a -nix variant, and recent versions of Windows include interfaces for interacting with the computer in a nix-y way.

Few things in the history of computing date 50 years back, and is still used because there are no viable alternatives. If one have something like the Bildung of the humanities in computing, time-less knowledge and skills, Unix would be the equivalent.

(4)
Understanding what Unix and The Unix Philosophy is, is to understand why it was made - its purposes, and uses.

Unix is a means to general-purpose computing, not a set of tools to solve a particular set of problems, but rather a set of tools to solve any problem – problems in general. Unix is the opposite of the monolithic application, the opposite of specialized software.

(5)
What are the dangers of specialization? How can general dumbness be smart? Well, it’s a smart kind of stupid. I believe that all kinds of specialization – if concentrated in one and only one area – faces the risk of tunnel vision, of being narrow-minded.

(6)
I want to demonstrate the smartness of dumbness – a special kind of dumbness – by contrasting the Unix way of solving problems with how problems are solved within a monolithic environment.

Let's imagine the need to, from weather data, produce some visualization, using Microsoft Excel.

(7)
First, we would copy the data from some kind of database, API, data source. Then we paste the data into Excel, and format it according to our needs.

(8)
We select the data to be visualized, and select between a set of options.

(9)
After we've decided how to view the data, we produce a visualization. Now we can save the visualization on our device.

(10)
But what if we wanted to automate this procedure? ...what if wanted to fetch all data on a regular basis (curl, cronjob), format it (sed, awk), produce a visualization (another tool), and save it (weaved together by a Bash script)?

This would not be possible within Excel, nor could Excel be expanded in such a manner if Microsoft did not make this a new feature.

We produced the visualization by simple means (highlighting, selecting, clicking), but parts of the procedure cannot be automated. We're locked-in in a clever solution for producing visualizations.

The smart design choices of Excel enabled us to quickly produce a visualization, but the same design choices trapped us in a monolith, unable to collaborate with for instance the browser (another monolith) from where we retrieved the data.

(11)
Understanding what Unix and The Unix Philosophy is, is to understand a background story and be inspired to create our own stories. Invented user stories, common use cases define what a machine or tool is. A monolith offers paths, a user is forced into a predefined set of paths. Of course Unix has limits, but the set of possible paths are in comparison endless, you can tell almost any story, given skills.

(12)
The Unix Philosophy can be expressed as a set of beliefs.

(13)
Unix environments prefer moving parts over fixed parts. Moving parts can be tinkered with. And tinkering is playing is exploring is means for discovery. Since Unix environments are hackable by default and this creates freedom to express a large set of computations, tasks, user stories.

(14)
Tools in Unix are written to do one thing, and do it well. They are built as modules in one large eco-system whose parts can be composed like general-purpose lego.

This script was written by Douglas McIlroy, a computer scientist involved in the making of Unix, as a response to an application stretching 10 pages, written in Pascal. The application counts word frequency, and prints the most used word along with the frequency.

McIlroys script uses *tr* to firstly separate words by a new line, then *tr* again to make all words lower case, sorts them (*sort*), keeping only unique elements in the list (along with the frequency, *uniq -c*), sorts the result in descending order, and prints the first line using *sed*. Bam! Done.

This application holds no state; in fact, it is purely functional, making it robust and immune to possible bugs. Making the same program in a 'common' programming language would be way more complicated.

The simplicity of McIlroys script, is made possible by a collaborative interface, an environment in which tools can be composed, and combined, without data loss or complexities of conversions.

Unix favor collaborative interfaces over the competitive interfaces of monolithic applications, enclosed in their own logic with (at best) some opaque window showing an almost desolate landscape.

(15)
Modularity, simplicity, and openness all share common properties. Exaggerated, Lego comes in two shapes – a type which general-purpose, and a type which is narrow. Both have limits in how they can be connected to other parts, but the possibilities of general-purpose allows you to build pretty much anything whereas a door (like the one here) surely allow you to connect it to other parts (and attach it to a wall, etc), but the set of possibilities is limited by the specificity of the shape. The door is a "specialized" form, so to speak.

(16)
Modularity, simplicity, and openness are linked in the Unix environment, and made possible by the maxim of text as a universal interface. All tools are written to take plain text - either by piping, standard output or from file streams - make some kind of computation, and spit it back to the standard out. Such tools are referred to as filters.

Pipes (and filters) are the glue that makes relationships between tools possible. The bread and butter of Unix.

To connect modules they must fit, be simple like lego, and have open interfaces.

(17)
While Unix tools often are written in languages such as C for efficiency and portability, Bash scripts solve tasks quickly (often using non-scripted tools). Some are made to be saved, filling purposes for a specific workflow and task, while other are written to be thrown away after being run once.

(18)
Text as a universal interface requires text to be stored in plain text, not in binary, or some other format. Even though parsing may be needed, depending on the format, no efforts need to be made to convert the data. This makes computations much easier. It lowers the hinders for exploring data, thus enabling discoveries.

(19)
The means for composing require familiarity between tools (POSIX compliance); tools must be made in the same form (following conventions for flags etc.), and all manuals follow the same format (man pages).

(20)
Tools built to be composed cannot be delicate, sensitive and fragile, but must be disciplined and streamlined according to standards and a heuristics claiming that things sooner or later will go wrong. Error handling follows a pattern, and error messages are encoded following a standard.

(21)
If no friction occurs when using a tool, no communication should be present. Outputting unnecessary text makes collaboration and piping difficult.

Suppose that after completing a task, a tool outputted "task completed". If this was piped, a intermediate pipe would be need using for instance sed, just for the purpose of removing "task completed", keeping only the result of the task performed.

(22)
The costs of using Unix tools shine through when we merely wanted to generate a single visualization of the weather data, and did not beforehand know about the Unix tools needed to pipe the data do a visualization tool.

(23)
Unix is as strong as the tools at hand, and the skill, and the creativity of the user.