Technological foundations of software development

Master your working environment

ICM – Computer Science Major – Course unit on Technological foundations of computer science
M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development
Maxime Lefrançois https://maxime-lefrancois.info
online: https://ci.mines-stetienne.fr/cps2/course/tfsd/

Objectives of the session

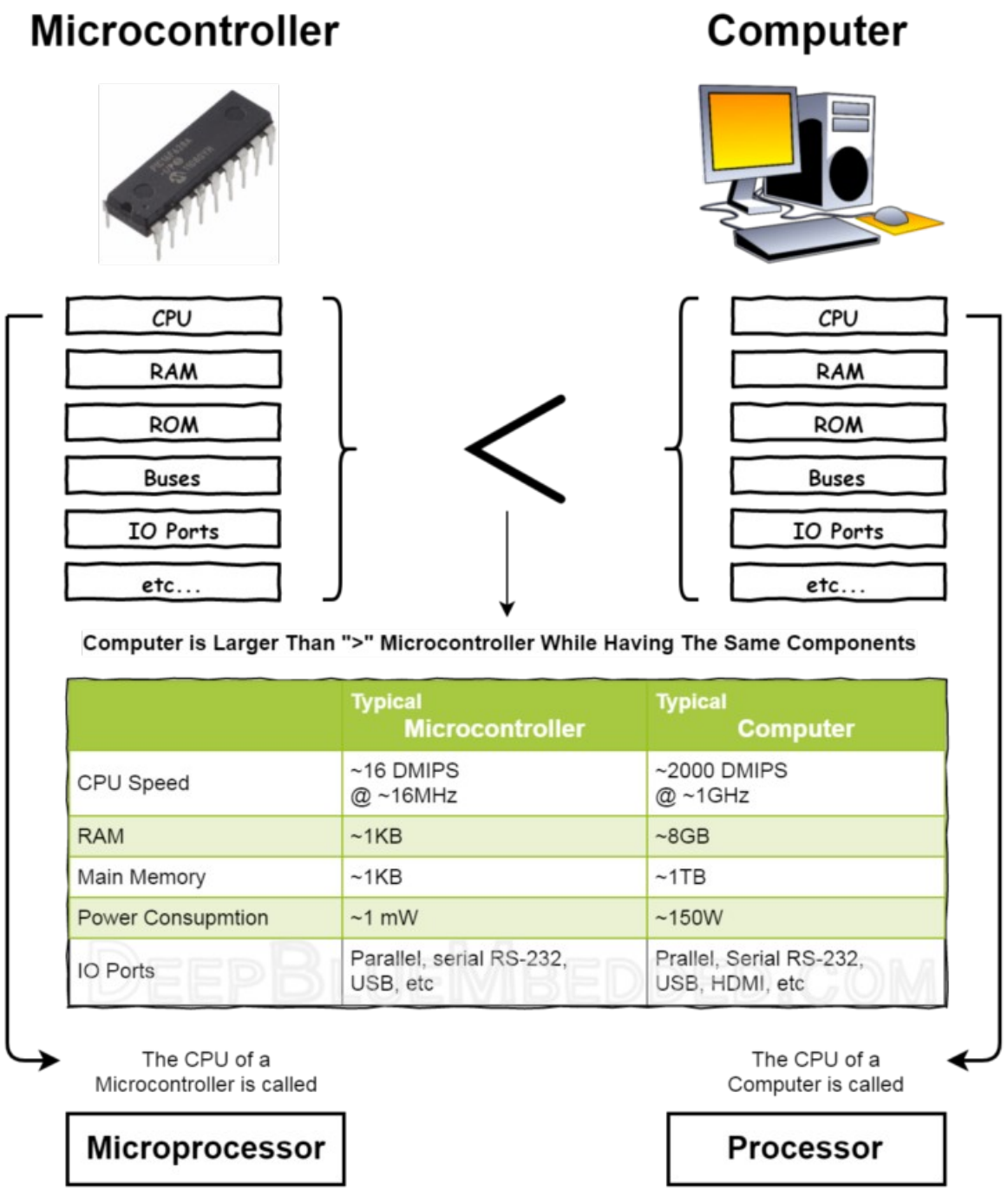
Ensure that you are familiar with your computer, your operating system, and the shell-like command-line programming environment

Reminders – computer ?

Programmable information processing system

Typically contains:

- CPU,
- RAM,
- ROM,
- Peripherals,
- IO Ports



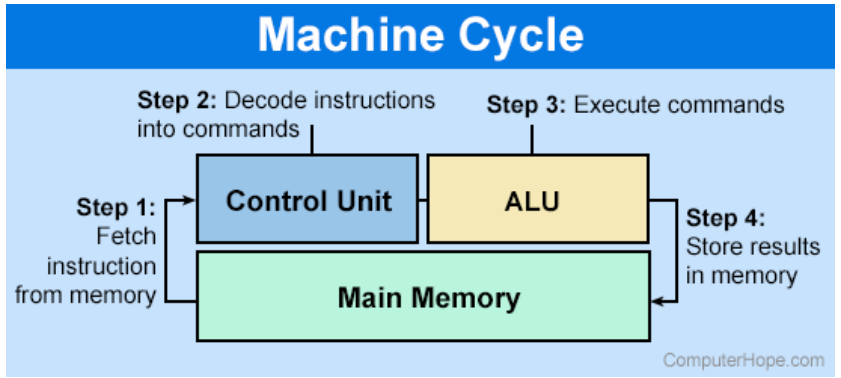
	Typical Microcontroller	Typical Computer
CPU Speed	~16 DMIPS @ ~16MHz	~2000 DMIPS @ ~1GHz
RAM	~1KB	~8GB
Main Memory	~1KB	~1TB
Power Consupmtion	~1 mW	~150W
IO Ports	Parallel, serial RS-232, USB, etc	Prallel, Serial RS-232, USB, HDMI, etc

https://deepbluembedded.com/what-is-the-difference-between-microprocessor-and-microcontroller/

CPU

The central processing unit (CPU) or processor, is the unit which performs most of the processing inside a computer. It processes all instructions received by software running on the PC and by other hardware components, and acts as a powerful calculator.

https://www.techopedia.com/definition/2851/central-processing-unit-cpu



https://www.computerhope.com/jargon/a/alu.htm

A control unit (CU) handles all processor control signals. It directs all input and output flow, fetches code for instructions from microprograms and directs other units and models by providing control and timing signals. A CU component is considered the processor brain because it issues orders to just about everything and ensures correct instruction execution.

https://www.techopedia.com/definition/2855/control-unit-cu

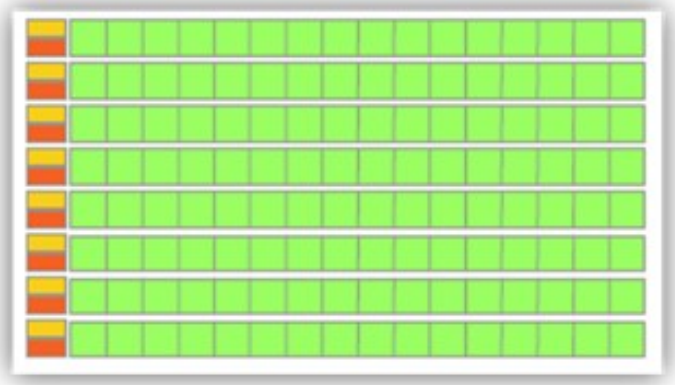
An arithmetic logic unit (ALU) is a major component of the central processing unit of a computer system. It does all processes related to arithmetic and logic operations that need to be done on instruction words.

https://www.techopedia.com/definition/2849/arithmetic-logic-unit-alu

CPU

* Low compute density
* Complex control logic
* Large caches (L1$/L2$, etc.)
* Optimized for serial operations
  - Fewer execution units (ALUs)
  - Higher clock speeds
* Shallow pipelines (<30 stages)
* Low Latency Tolerance
* Newer CPUs have more parallelism

GPU



* High compute density
* High Computations per Memory Access
* Built for parallel operations
  - Many parallel execution units (ALUs)
  - Graphics is the best known case of parallelism
* Deep pipelines (hundreds of stages)
* High Throughput
* High Latency Tolerance
* Newer GPUs:
  - Better flow control logic (becoming more CPU-like)
  - Scatter/Gather Memory Access
  - Don't have one-way pipelines anymore

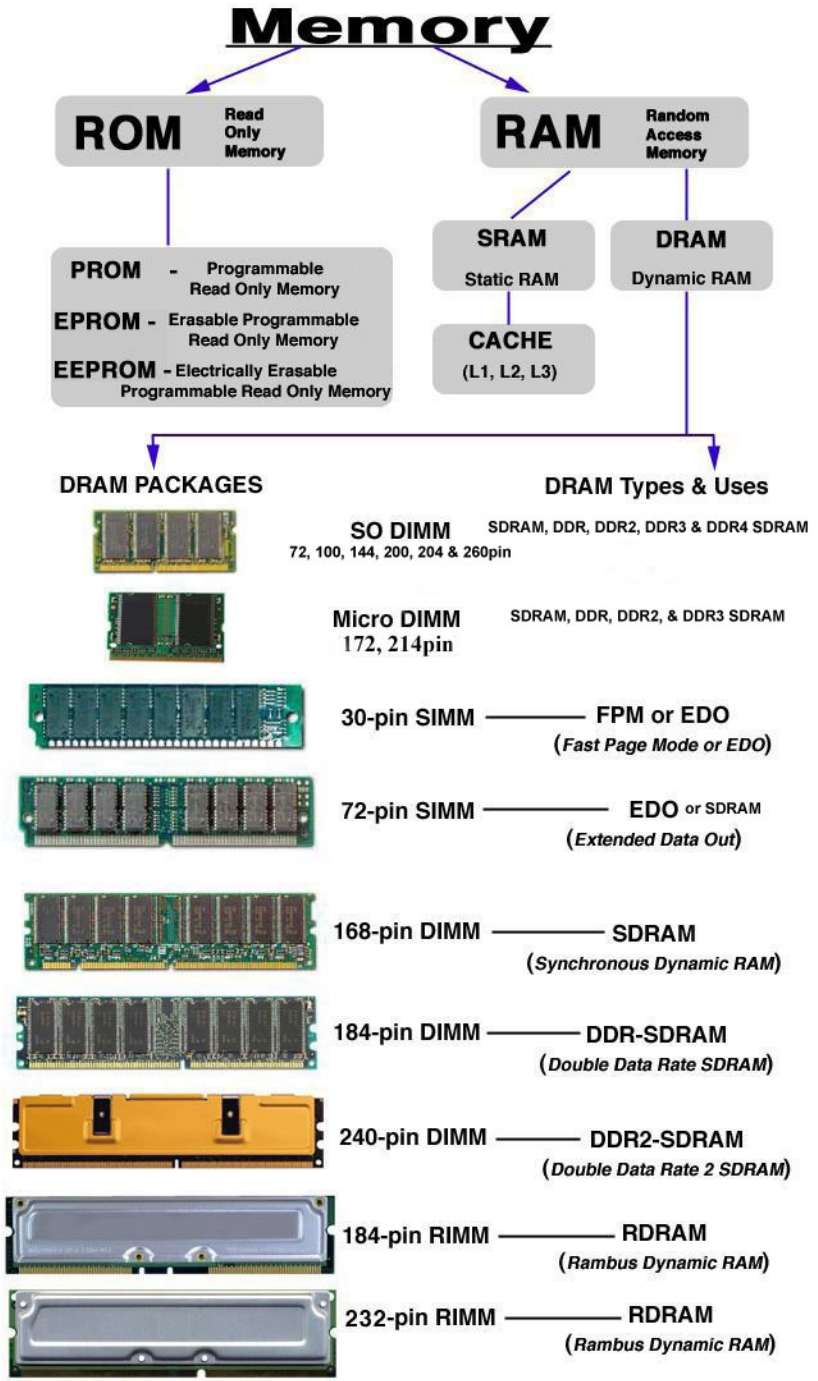
https://www.electronicspecifier.com/products/communications/vivante-es-design-magazine-gpus-the-next-must-have

RAM vs ROM

RAM	ROM
1. Temporary Storage.	1. Permanent storage.
2. Store data in MBs.	2. Store data in GBs.
3. Volatile.	3. Non-volatile.
4.Used in normal operations.	4. Used for startup process of computer.
5. Writing data is faster.	5. Writing data is slower.

Difference between RAM and ROM

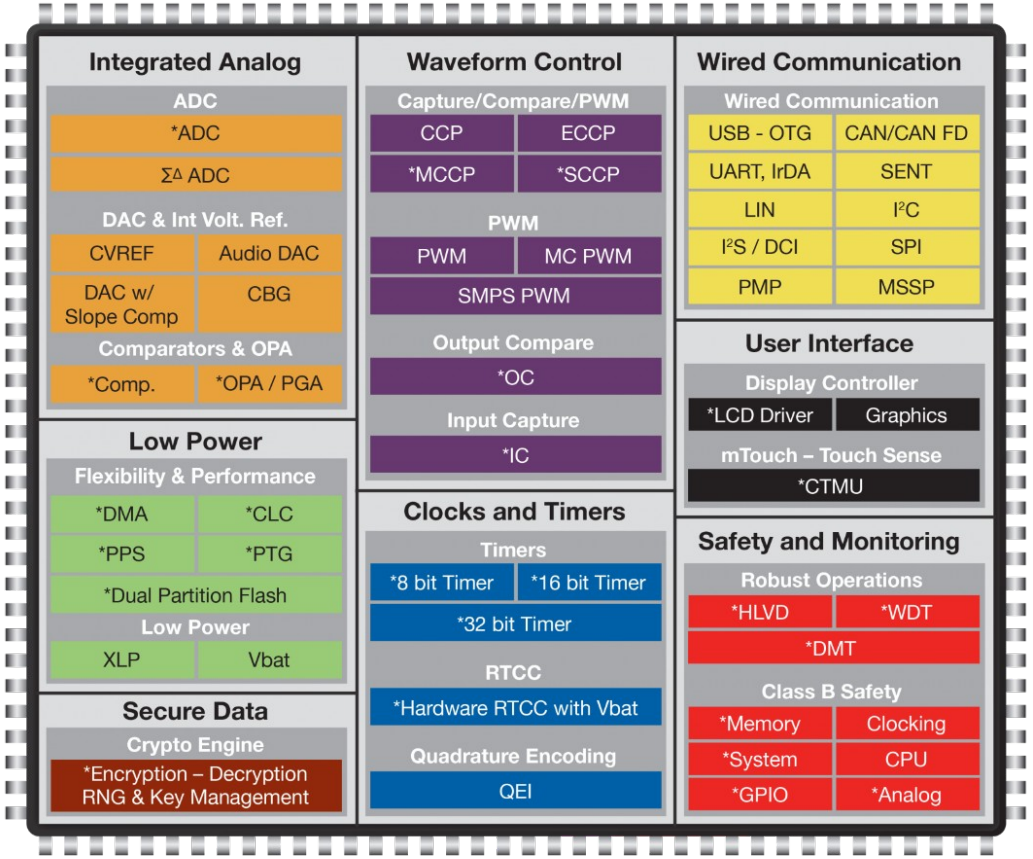
https://www.geeksforgeeks.org/random-access-memory-ram-and-read-only-memory-rom/



https://www.escotal.com/memory.html

Peripherals and I/O ports

Embedded peripherals



*Core Independent Peripherals (CIPs)

Example: Peripheral embedded in a PIC24 microcontroller
https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/16-bit-mcus/peripherals#

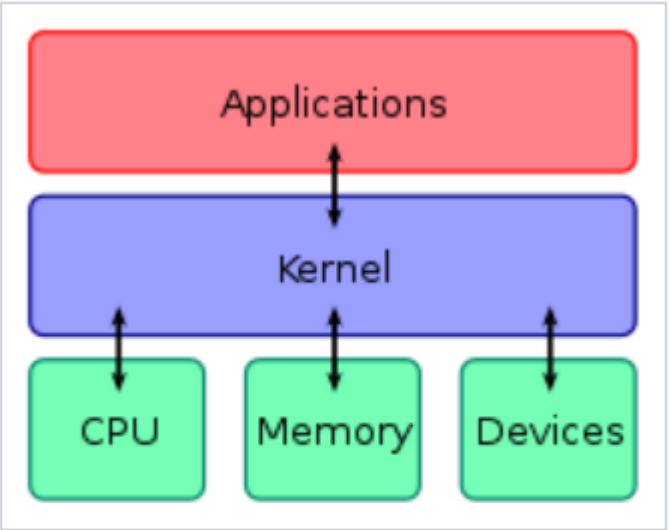
External peripherals

- Input, Output, Storage, Processing

I/O ports

Operating system- definition

A set of programs that directs the use
of a computer's resources
by application software.



A kernel connects the application software to the hardware of a computer.

https://en.wikipedia.org/wiki/Operating_system

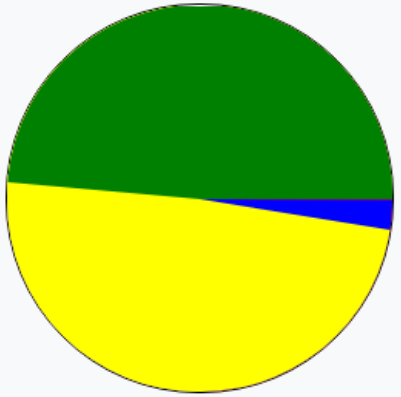
Brian L. Stuart, Principles of Operating Systems: Design & Applications, Cengage Learning EMEA, 2008 (ISBN 978-1418837693)

Operating system – usage share

Desktop/Laptop operating system browsing statistics

Windows	76.56%
macOS	17.1%
Unknown	2.68%
Linux	1.93%
Chrome OS	1.72%
FreeBSD	<0.01%

Desktop OS market share according to StatCounter for December 2020.[4]
Chrome OS is also based on the Linux kernel.

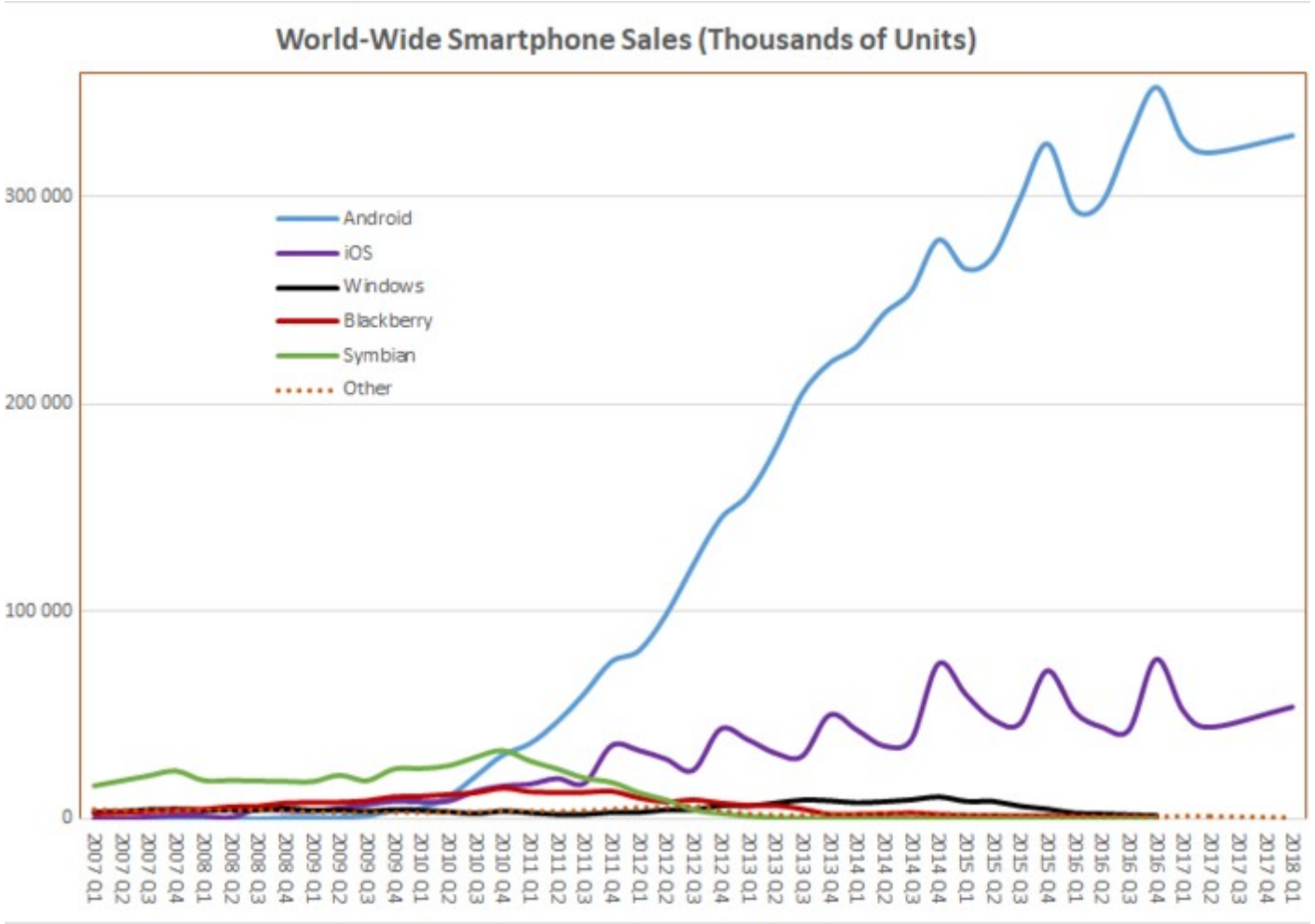


Since mid-2020, the world uses mobile computers more than desktop (including laptop) computers.[85][86] This is so every day of the week worldwide, though outside Asia and Africa, it is not true on weekdays. In all countries, mobile usage increases on weekends; in the United States, to a 54% share.[87][88][89][90] The proportions do vary widely by region (more so than by the day), e.g. in Africa desktop is at 38.23% while in South America it's at 66.67%, and in the United States at 51.34% (and similar in Europe). Considering all personal computing devices, Microsoft Windows is well below 50% usage share on every continent except South America (while it does on weekends), but Android got temporarily more popular Windows there; and Windows dipped to 45% in April 2020, likely due to COVID-19.[91]

- Smartphones (48.59%)
- Desktops/Laptops (48.85%)
- Tablets (2.5%)
- Console (0.06%)

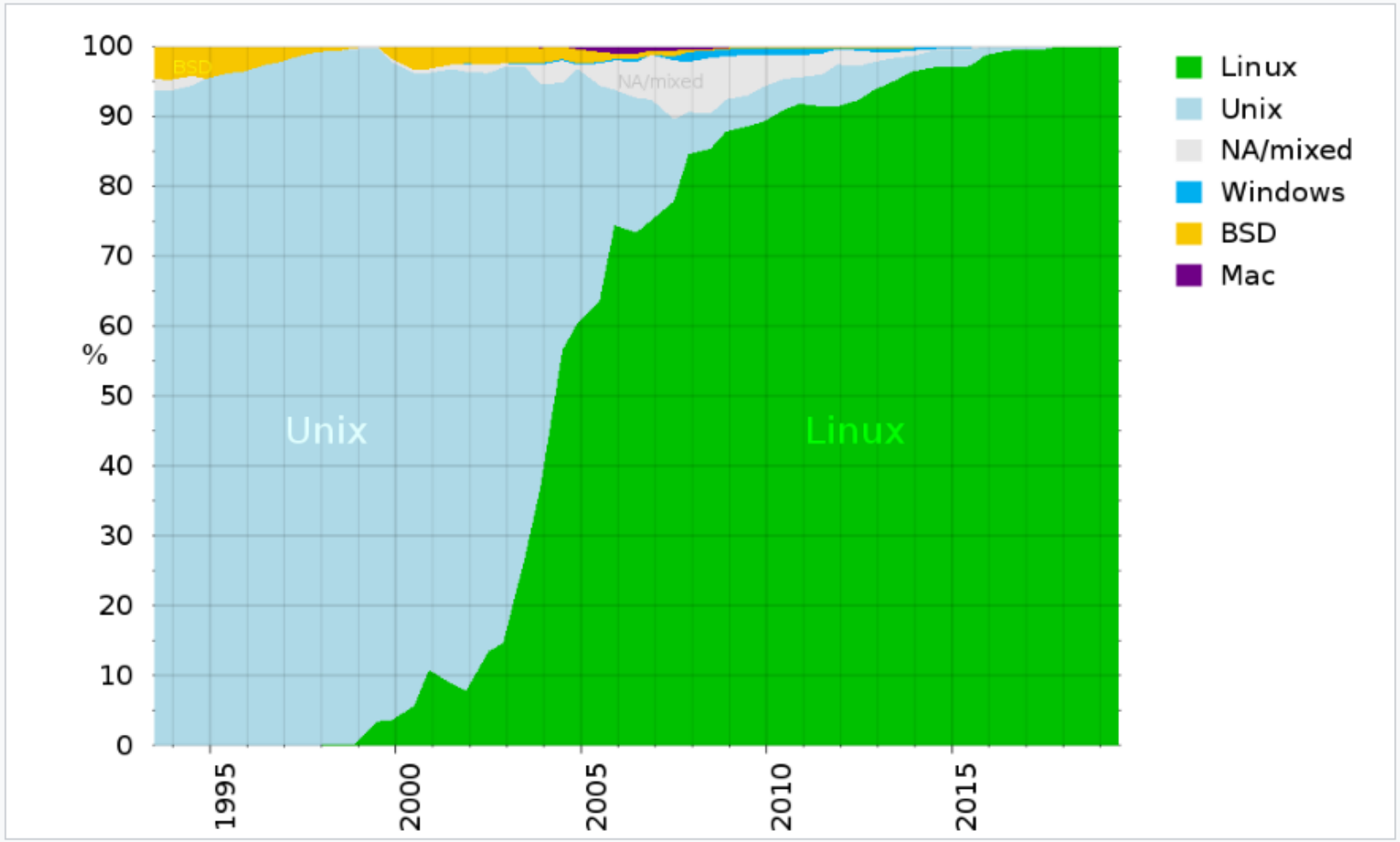
https://en.wikipedia.org/wiki/Usage_share_of_operating_systems

Operating system – smartphones



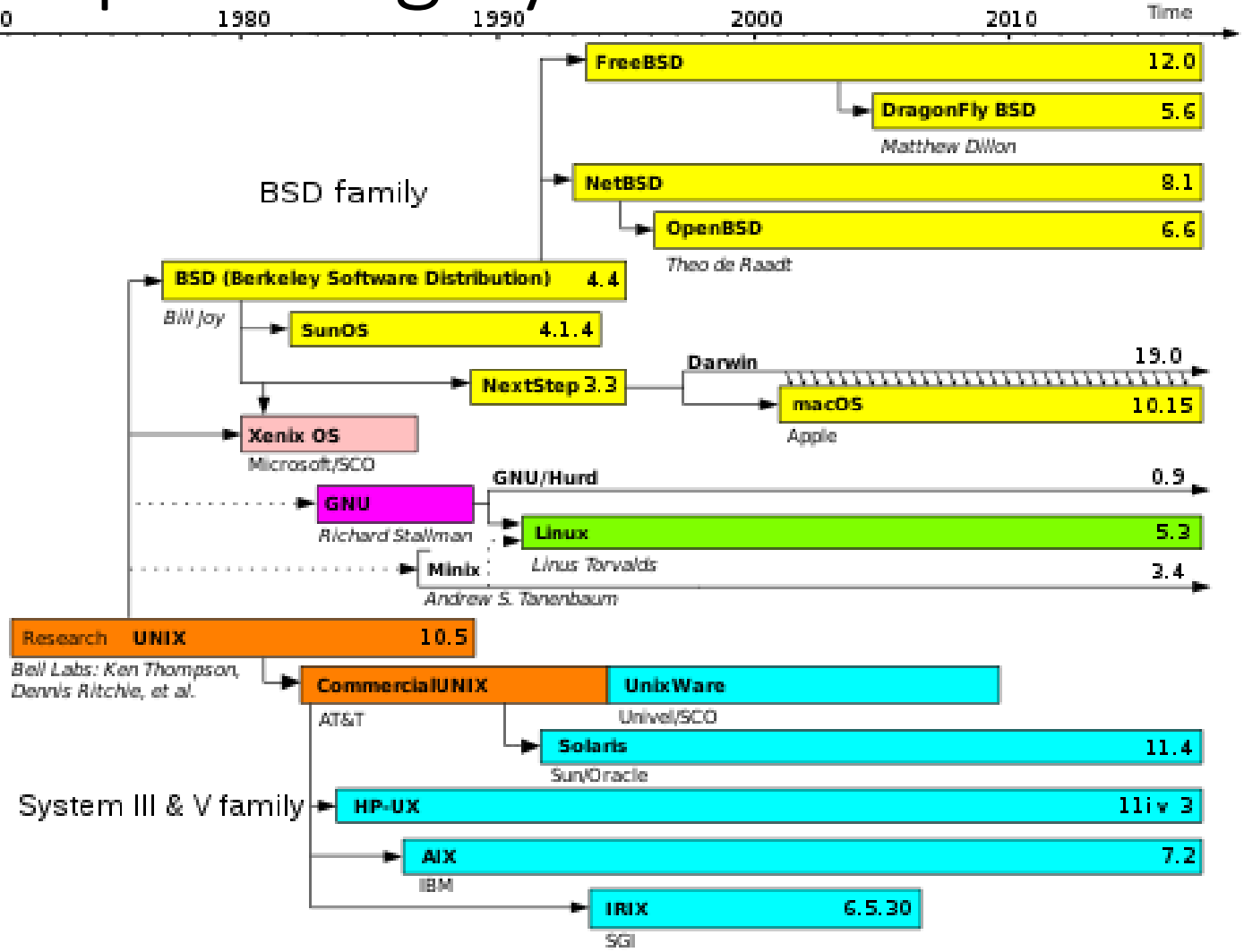
https://en.wikipedia.org/wiki/Usage_share_of_operating_systems

Operating system – supercomputers



Supercomputer OS family – 1993–2019 systems share according to TOP500[247]

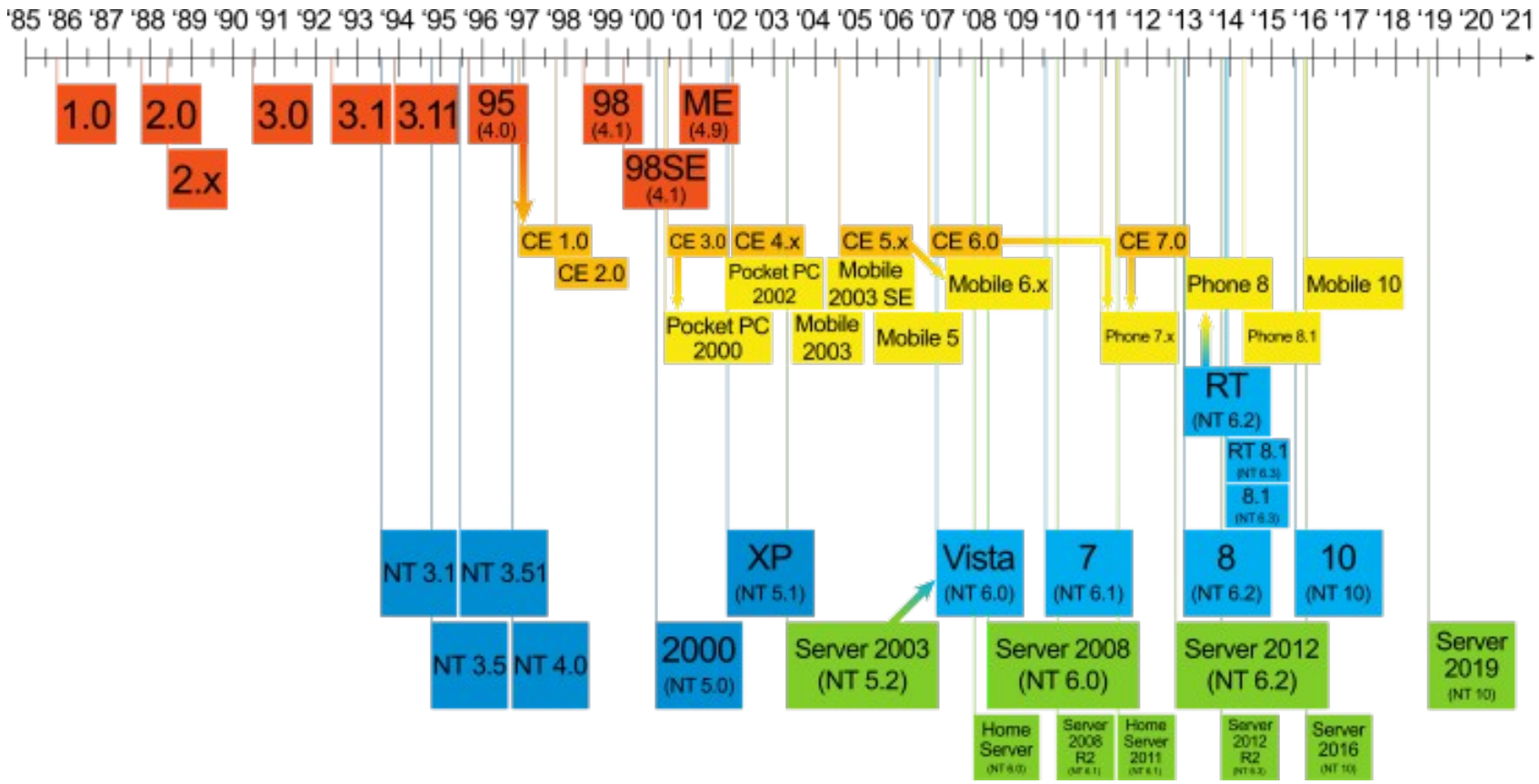
Your Operating System – Unix-like OS



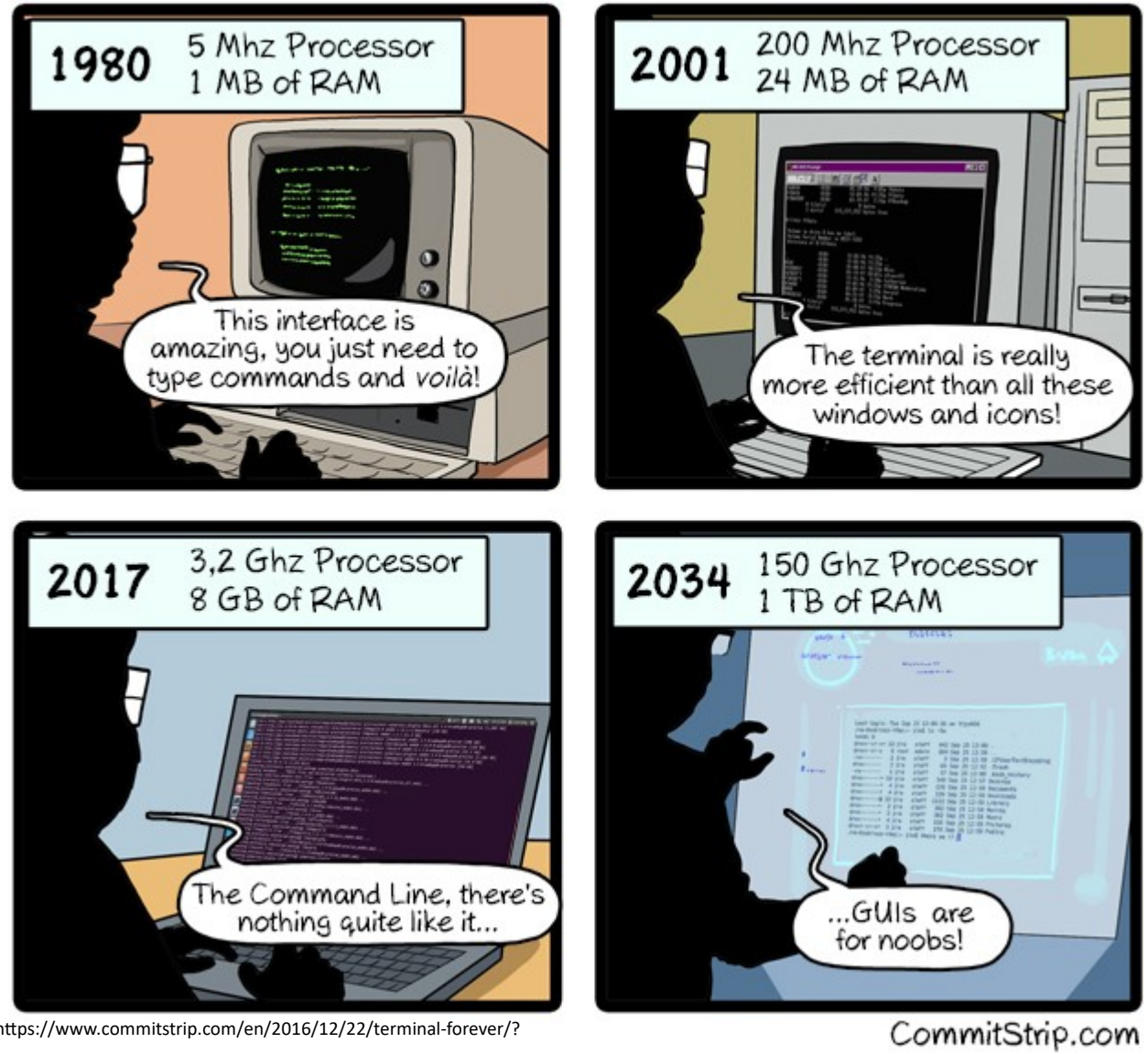
Simplified history of Unix-like operating systems. Linux shares similar architecture and concepts (as part of the POSIX standard) but does not share non-free source code with the original Unix or MINIX.

https://en.wikipedia.org/wiki/Linux

Your Operating System – Windows



Shell – Console – Terminal



https://www.commitstrip.com/en/2016/12/22/terminal-forever/?

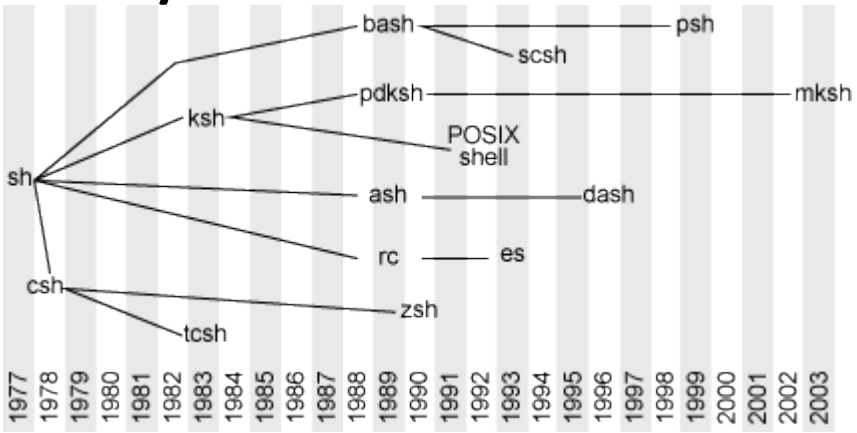
Shell – Console – Terminal

terminal = text input/output environment

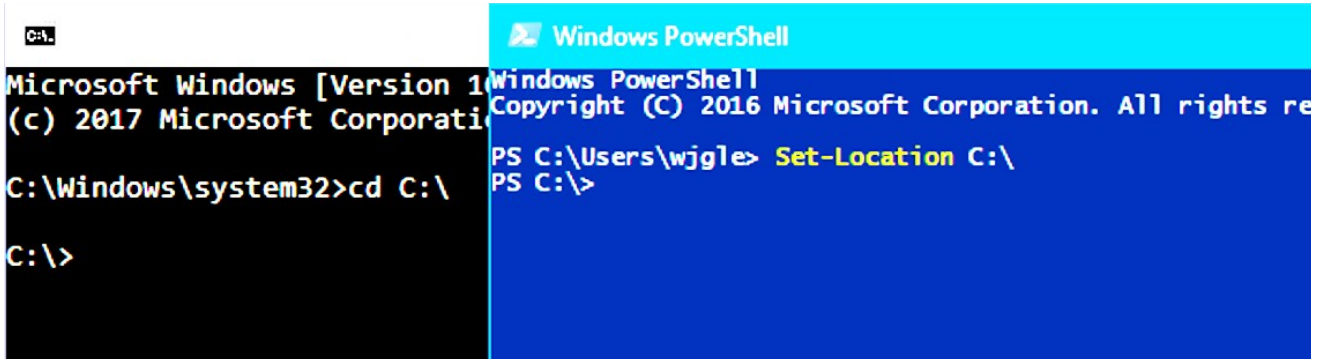
console = physical terminal

shell = command line interpreter - software layer that provides the user interface of an operating system

Unix / Linux



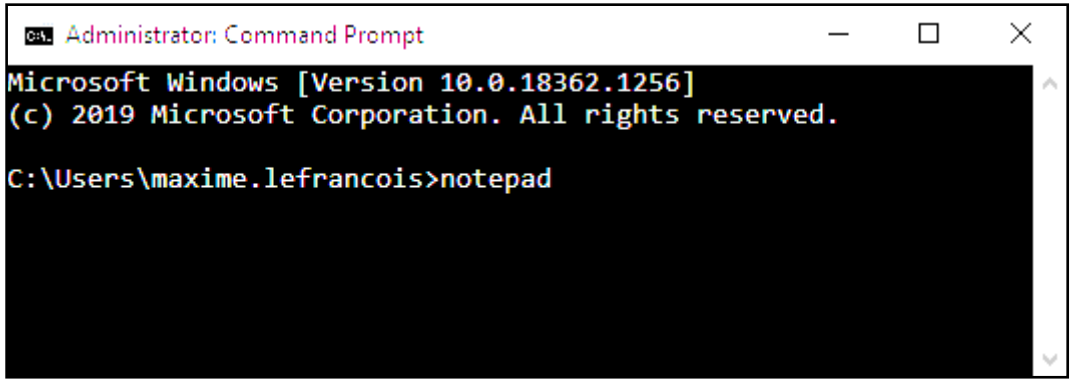
Windows



1987

2006

Run a program in the console



Run a program in the console

```
mlefranc@FAYOL-LEFRANCOIS:/mnt/c/Users/maxime.lefrancois$ vim
```

```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.18362.1256]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\maxime.lefrancois>notepad
```

Where were these programs found ?

```
mlefranc@FAYOL-LEFRANCOIS:/mnt/c/Users/maxime.lefrancois$ which vim
/usr/bin/vim
mlefranc@FAYOL-LEFRANCOIS:/mnt/c/Users/maxime.lefrancois$
```

```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.18362.1256]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\maxime.lefrancois>where notepad
C:\Windows\System32\notepad.exe
C:\Windows\notepad.exe

C:\Users\maxime.lefrancois>
```

Programs location

Some programs are integrated in the Shell, the others are searched in the file system by scanning the PATH environment variable

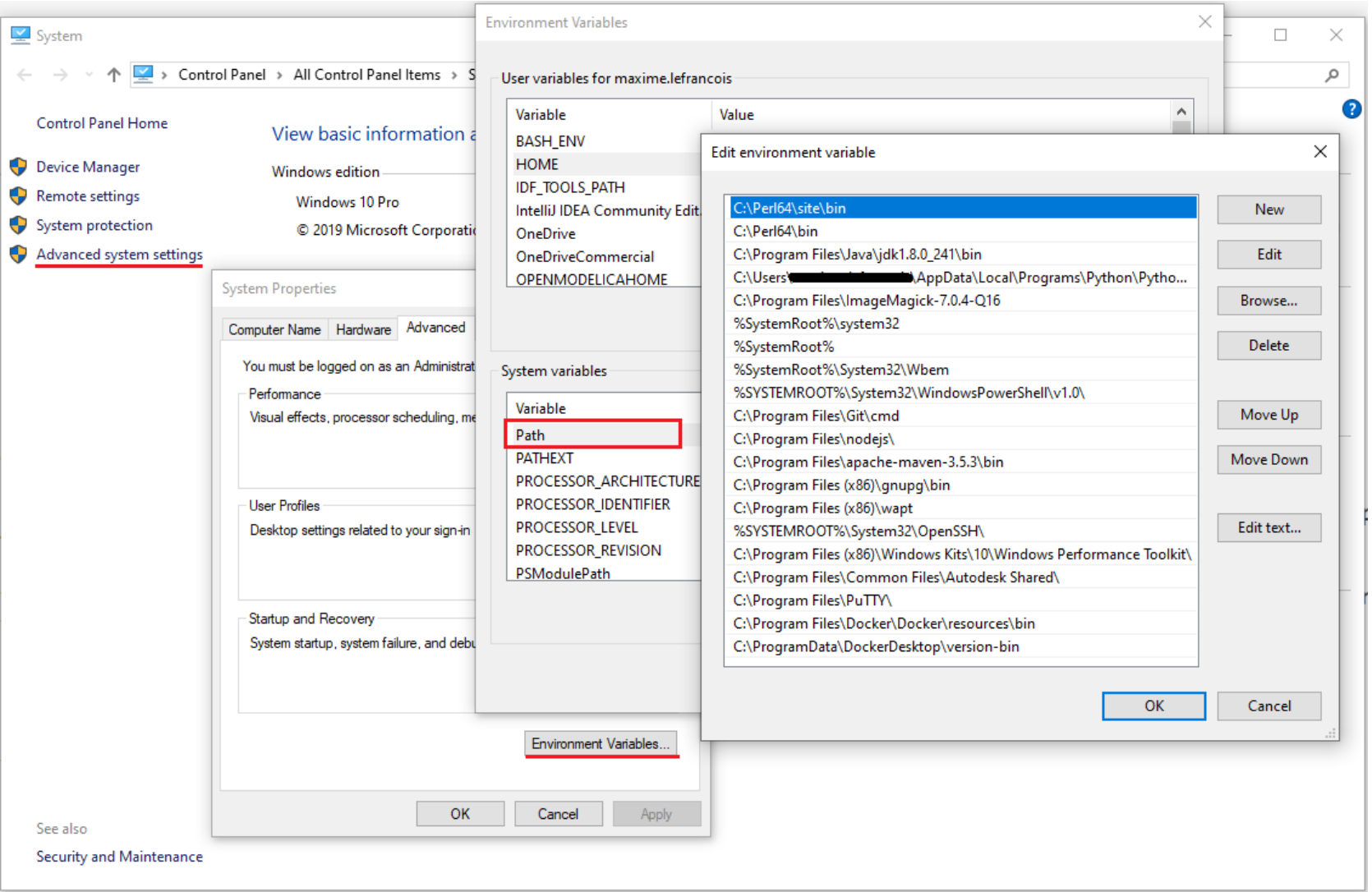
Administrator: Command Prompt

```
C:\Users\maxime.lefrancois>where mkdir
INFO: Could not find files for the given pattern(s).

C:\Users\maxime.lefrancois>where notepad
C:\Windows\System32\notepad.exe
C:\Windows\notepad.exe

C:\Users\maxime.lefrancois>echo %PATH%
C:\Perl64\site\bin;C:\Perl64\bin;C:\Program Files\Java\jdk1.8.0_241\bin;C:\Users\maxime.lefrancois\AppData\Local\Program
s\Python\Python39\Scripts;C:\Program Files\ImageMagick-7.0.4-Q16;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem
;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Program Files\Git\cmd;C:\Program Files\nodejs\;C:\Program Files\apache-m
aven-3.5.3\bin;C:\Program Files (x86)\gnupg\bin;C:\Program Files (x86)\wapt;C:\WINDOWS\System32\OpenSSH\;C:\Program File
s (x86)\Windows Kits\10\Windows Performance Toolkit\;C:\Program Files\Common Files\Autodesk Shared\;C:\Program Files\PuT
TY\;C:\Program Files\Docker\Docker\resources\bin;C:\ProgramD
Data\Local\Programs\Python\Python39\Scripts\;C:\Users\maxime
\maxime.lefrancois\.cargo\bin;C:\texlive\2016\bin\win32;C:\U
rs\maxime.lefrancois\AppData\Local\Microsoft\WindowsApps;C:\
\JetBrains\IntelliJ-IDEA-CE-2018.3.3\bin;C:\Users\maxime.lef
 (x86)\Nmap;C:\Program Files\JetBrains\IntelliJ IDEA Communi
```

mlefranc@FAYOL-LEFRANCOIS:

```
mlefranc@FAYOL-LEFRANCOIS:/mnt/c/Users/maxime.lefrancois$ type cd
cd is a shell builtin
mlefranc@FAYOL-LEFRANCOIS:/mnt/c/Users/maxime.lefrancois$ type mkdirmkdir is h
ashed (/usr/bin/mkdir)
mlefranc@FAYOL-LEFRANCOIS:/mnt/c/Users/maxime.lefrancois$ echo $PATH/usr/local
/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
```

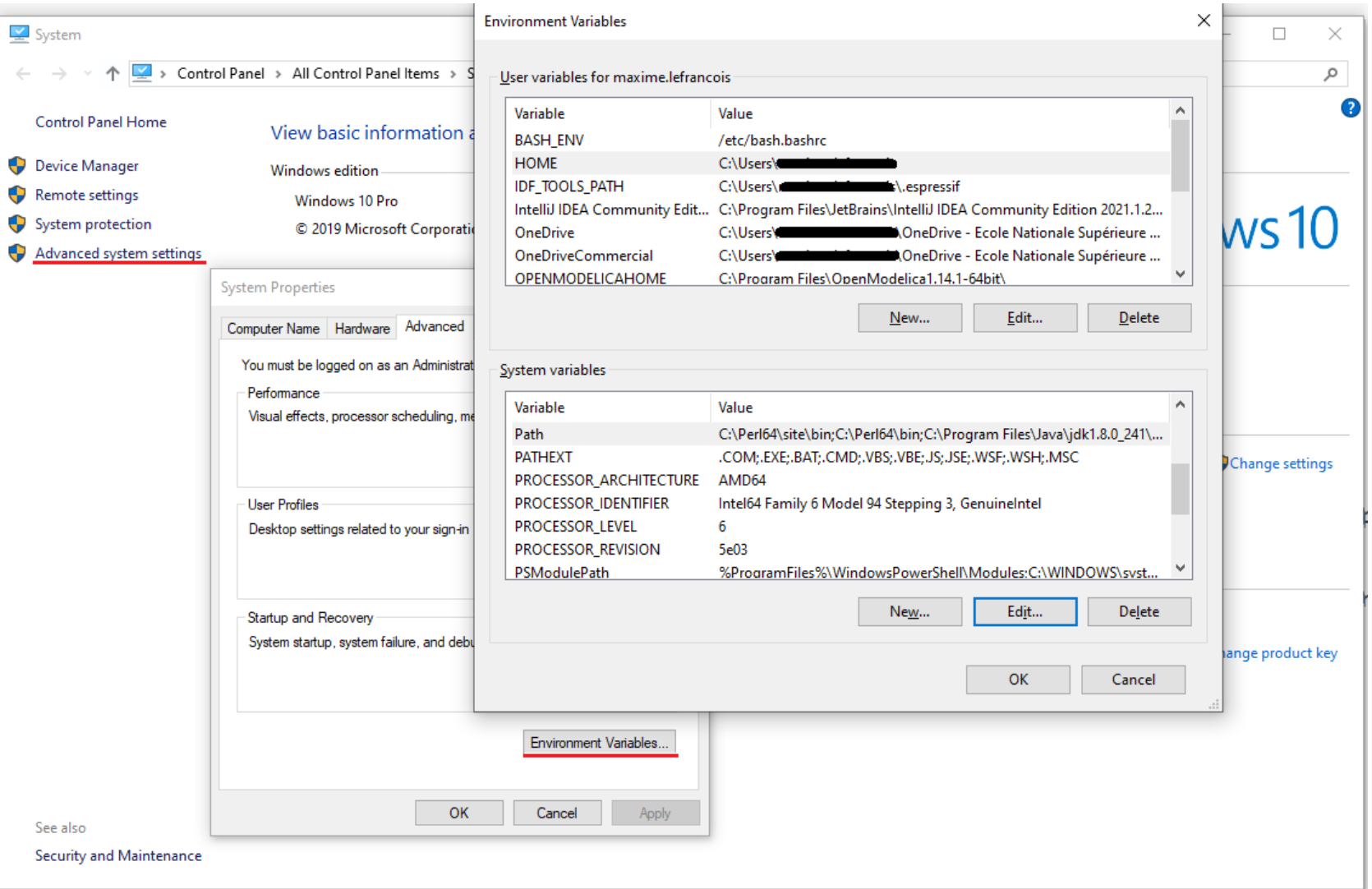
Windows – Environment variable **%PATH%**



%PATH% contains a list of files. When a command is executed in the CMD prompt:

1. Windows first looks for an executable file in the current directory.
2. If this fails, it scans %PATH% to find it.

Windows – Other environment variable



In cmd.exe:

Display a variable:
> echo %HOME%

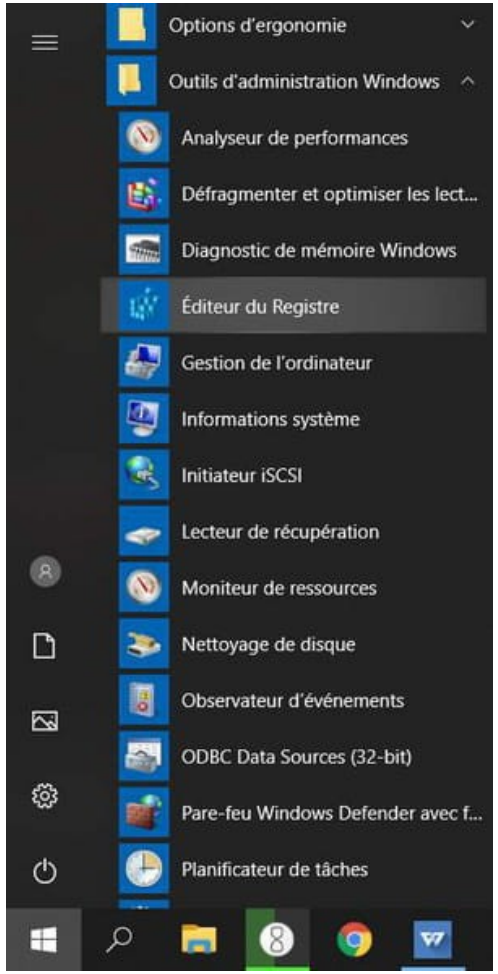
Display all variables:
> set

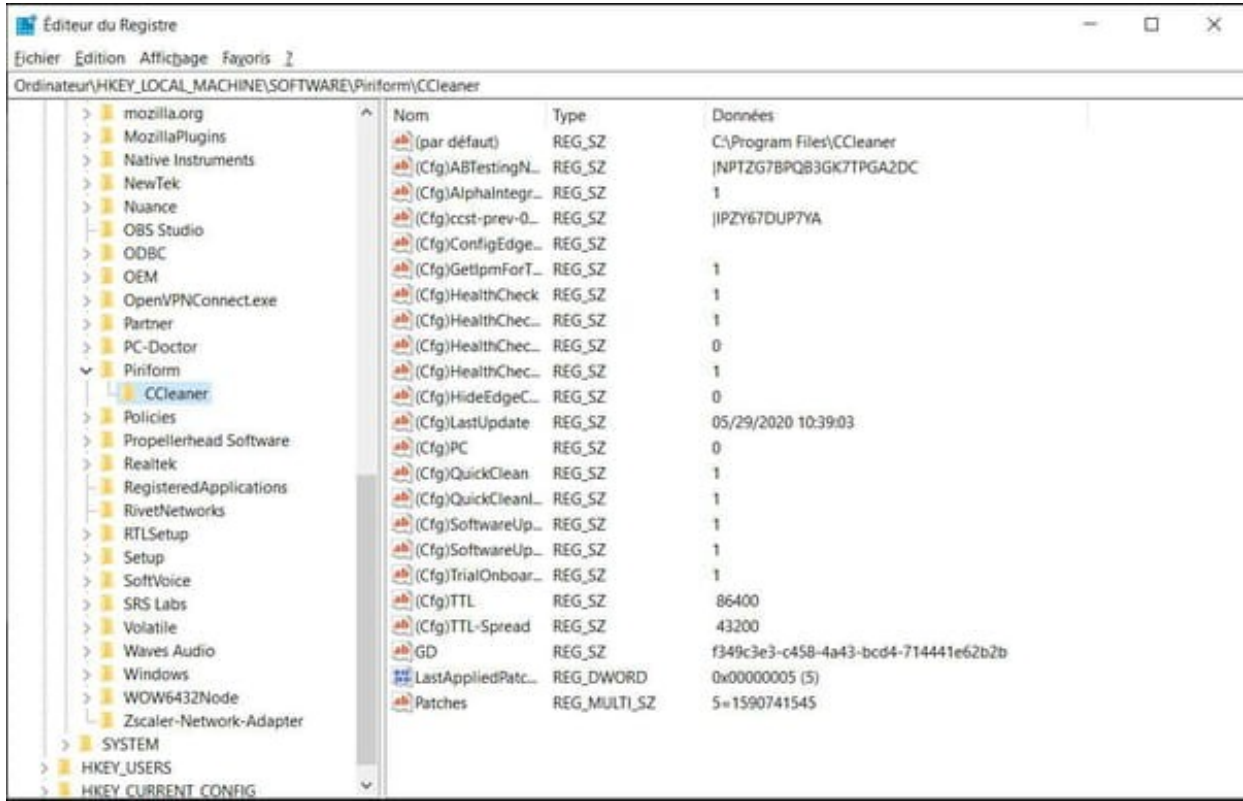
Change a variable
(for the current session)
> set HOME=hello world

Windows – Registry

Registre Windows

The registry is a database used by the Windows operating system. It contains the configuration data of the operating system and other installed software that want to use it.
Environment variables are stored in the registry.





More information with ex. https://learn.microsoft.com/en-us/troubleshoot/windows-server/performance/windows-registry-advanced-users

Linux – Different configurations depending on the Shell

*We will only see **bash** (Bourne shell, default in Ubuntu) and **zsh** (Z shell, default in iOS from Catalina)*

Next: Interactive Shells, Previous: Invoking Bash, Up: Bash Features [Contents][Index]

6.2 Bash Startup Files

This section describes how Bash executes its startup files. If any of the files exist but cannot be read, Bash reports an error. Tildes are expanded in filenames as described above under Tilde Expansion (see Tilde Expansion).

Interactive shells are described in Interactive Shells.

Invoked as an interactive login shell, or with `--login`

When Bash is invoked as an interactive login shell, or as a non-interactive shell with the `--login` option, it first reads and executes commands from the file `/etc/profile`, if that file exists. After reading that file, it looks for `~/.bash_profile`, `~/.bash_login`, and `~/.profile`, in that order, and reads and executes commands from the first one that exists and is readable. The `--noprofile` option may be used when the shell is started to inhibit this behavior.

When an interactive login shell exits, or a non-interactive login shell executes the `exit` builtin command, Bash reads and executes commands from the file `~/.bash_logout`, if it exists.

https://www.gnu.org/software/bash/manual/html_node/Bash-Startup-Files.html

Z shell Startup Files

There are five startup files that zsh will read commands from:

```
$ZDOTDIR/.zshenv
$ZDOTDIR/.zprofile
$ZDOTDIR/.zlogin
$ZDOTDIR/.zlogout
```

If `ZDOTDIR` is not set, then the value of `HOME` is used; this is the usual case.

`.zshenv' is sourced on all invocations of the shell, unless the `-f` option is set. It should contain commands to set the command search path, plus other important environment variables. `.zshenv' should not contain commands that produce output or assume the shell is attached to a tty.

`.zshrc' is sourced in interactive shells. It should contain commands to set up aliases, functions, options, key bindings, etc.

`.zlogin' is sourced in login shells. It should contain commands that should be executed only in login shells. `.zlogout' is sourced when login shells exit. `.zprofile' is similar to `.zlogin', except that it is sourced before `.zshrc'.

https://zsh.sourceforge.io/Intro/intro_3.html (modified slightly)

Linux – Different configurations depending on the Shell

*We will only see **bash** (Bourne shell, default in Ubuntu) and **zsh** (Z shell, default in iOS from Catalina)*

Next: Interactive Shells, Previous: Invoking Bash, Up: Bash Features [Contents][Index]

6.2 Bash Startup Files

This section describes how Bash executes its startup files. If any of the files exist but cannot be read, Bash reports an error. Tildes are expanded in filenames as described above under Tilde Expansion (see Tilde Expansion).

Interactive shells are described in Interactive Shells.

Invoked as an interactive login shell, or with `--login`

When Bash is invoked as an interactive login shell, or as a non-interactive shell with the `--login` option, it first reads and executes commands from the file `/etc/profile`, if that file exists. After reading that file, it looks for `~/.bash_profile`, `~/.bash_login`, and `~/.profile`, in that order, and reads and executes commands from the first one that exists and is readable. The `--noprofile` option may be used when the shell is started to inhibit this behavior.

When an interactive login shell exits, or a non-interactive login shell executes the `exit` builtin command, Bash reads and executes commands from the file `~/.bash_logout`, if it exists.

https://www.gnu.org/software/bash/manual/html_node/Bash-Startup-Files.html

mlefranc@FAYOL-LEFRANCOIS:

```
mlefranc@FAYOL-LEFRANCOIS:/mnt/c/Users/maxime.lefrancois$ cat ~/.profile
# ~/.profile: executed by the command interpreter for login shells.
# This file is not read by bash(1), if ~/.bash_profile or ~/.bash_login
# exists.
# see /usr/share/doc/bash/examples/startup-files for examples.
# the files are located in the bash-doc package.

# the default umask is set in /etc/profile; for setting the umask
# for ssh logins, install and configure the libpam-umask package.
#umask 022

# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/.local/bin" ] ; then
    PATH="$HOME/.local/bin:$PATH"
fi
source "$HOME/.cargo/env"
```

Linux – Different configurations depending on the Shell

*We will only see **bash** (Bourne shell, default in Ubuntu) and **zsh** (Z shell, default in iOS from Catalina)*

mlefranc@FAYOL-LEFRANCOIS:

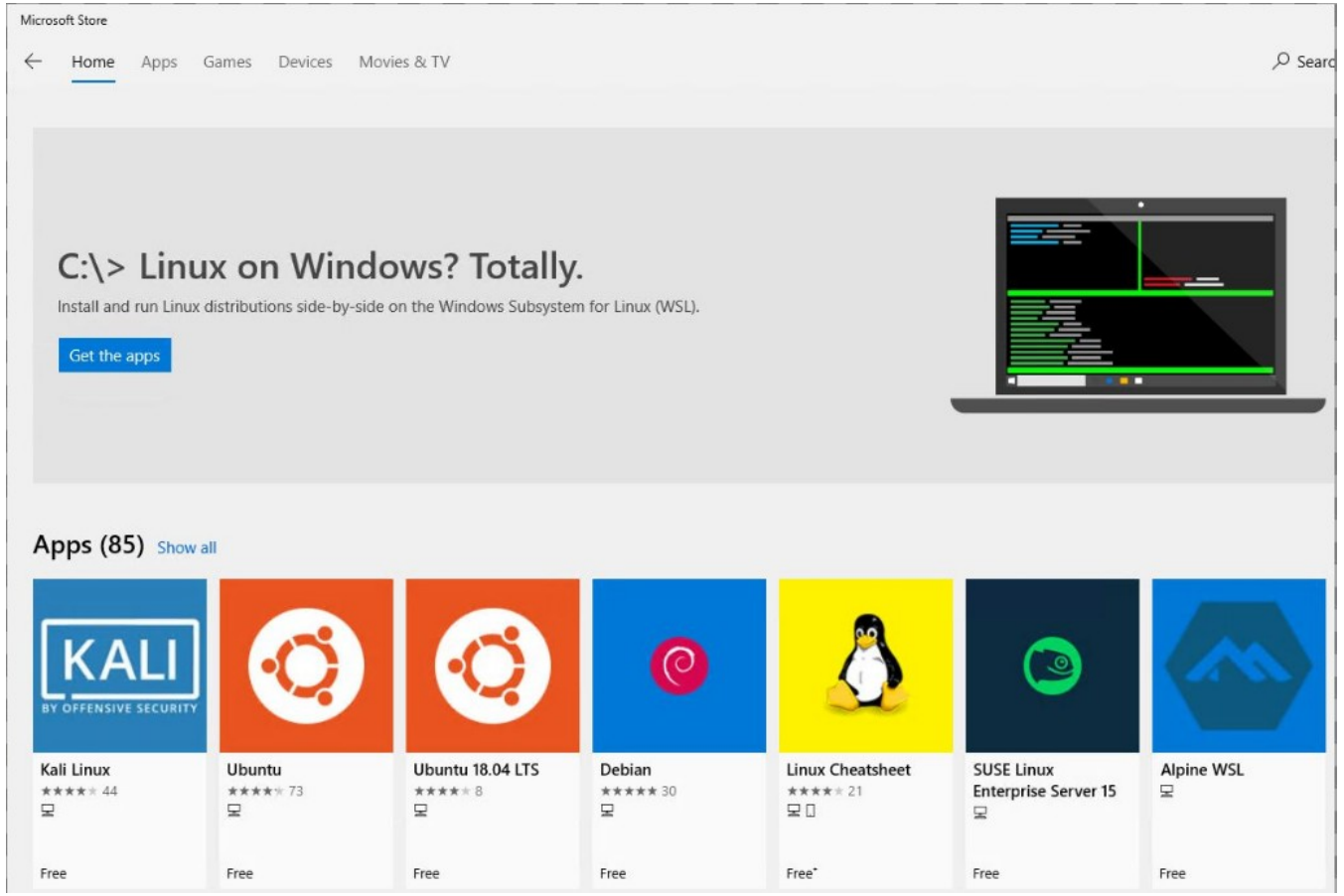
```
mlefranc@FAYOL-LEFRANCOIS:/mnt/c/Users/maxime.lefrancois$ tail -30 ~/.bashrc
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi

# git pgp https://stackoverflow.com/questions/41052538/git-error-gpg-failed-to
-sign-data/41054093
export GPG_TTY=$(tty)

# always have display option for guis
export DISPLAY=$(cat /etc/resolv.conf | grep nameserver | awk '{print $2}'):0
export LIBGL_ALWAYS_INDIRECT=1

# issue https://github.com/microsoft/WSL/issues/2855#issuecomment-613935650
#export LIBGL_ALWAYS_INDIRECT=0

export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64

# ssh-agent auto-launch (0 = agent running with key; 1 = w/o key; 2 = not run.
)
agent_run_state=$(ssh-add -l >| /dev/null 2>&1; echo $?)
if [ $agent_run_state = 2 ]; then
  eval $(ssh-agent -s)
  ssh-add ~/.ssh/id_rsa_ci_4096
elif [ $agent_run_state = 1 ]; then
  ssh-add ~/.ssh/id_rsa_ci_4096
fi
source "$HOME/.cargo/env"
export PATH=$PATH:~/apache-jena-4.1.0/bin/
```

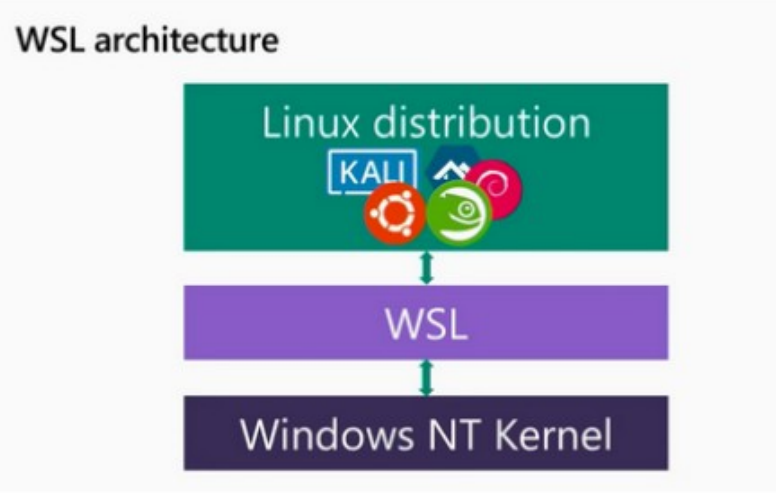
mlefranc@FAYOL-LEFRANCOIS:

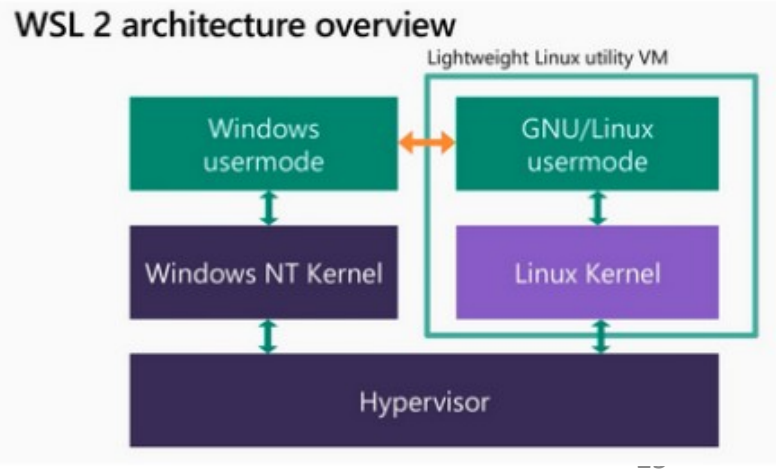
```
mlefranc@FAYOL-LEFRANCOIS:/mnt/c/Users/maxime.lefrancois$ cat ~/.profile
# ~/.profile: executed by the command interpreter for login shells.
# This file is not read by bash(1), if ~/.bash_profile or ~/.bash_login
# exists.
# see /usr/share/doc/bash/examples/startup-files for examples.
# the files are located in the bash-doc package.

# the default umask is set in /etc/profile; for setting the umask
# for ssh logins, install and configure the libpam-umask package.
#umask 022

# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/.local/bin" ] ; then
    PATH="$HOME/.local/bin:$PATH"
fi
source "$HOME/.cargo/env"
```

Windows Subsystem for Linux (WSL / WSL2)

Will allow you to use a UNIX/Linux kernel and a Shell, regardless of your OS







https://lecrabeinfo.net/installer-wsl-windows-subsystem-for-linux-sur-windows-10.html

Access to environment variables (e.g. with Java)

Environment Variables (The Java

docs.oracle.com/javase/tutorial/essential/environment/env.html

ORACLE Java Documentation

The Java™ Tutorials

The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available.
See Java Language Changes for a summary of updated language features in Java SE 9 and subsequent releases.
See JDK Release Notes for information about new features, enhancements, and removed or deprecated options for all JDK releases.

Environment Variables

Many operating systems use *environment variables* to pass configuration information to applications. Like properties in the Java platform, environment variables are key/value pairs, where both the key and the value are strings. The conventions for setting and using environment variables vary between operating systems, and also between command line interpreters. To learn how to pass environment variables to applications on your system, refer to your system documentation.

Querying Environment Variables

On the Java platform, an application uses `System.getenv` to retrieve environment variable values. Without an argument, `getenv` returns a read-only instance of `java.util.Map`, where the map keys are the environment variable names, and the map values are the environment variable values. This is demonstrated in the `EnvMap` example:

```
import java.util.Map;

public class EnvMap {
    public static void main (String[] args) {
        Map<String, String> env = System.getenv();
        for (String envName : env.keySet()) {
            System.out.format("%s=%s%n",
                              envName,
                              env.get(envName));
        }
    }
}
```


https://docs.oracle.com/javase/tutorial/essential/environment/env.html

Access to environment variables (e.g. with Python)

docs.python.org/3/library/os.html

Table of Contents

The method should only return a `str` or `bytes` object, with the preference being for `str`.

`os.getenv(key, default=None)`

Return the value of the environment variable *key* as a string if it exists, or *default* if it doesn't. *key* is a string. Note that since `getenv()` uses `os.environ`, the mapping of `getenv()` is similarly also captured on import, and the function may not reflect future environment changes.

On Unix, keys and values are decoded with `sys.getfilesystemencoding()` and `'surrogateescape'` error handler. Use `os.getenvb()` if you would like to use a different encoding.

Availability: Unix, Windows.

There are tutorials for other languages
example for C: https://www.gnu.org/software/libc/manual/html_node/Environment-Access.html
example for node.js: https://nodejs.org/en/learn/command-line/how-to-read-environment-variables-from-nodejs

Run a program in the console

Name of the program, then list of arguments separated by spaces

example: small program **count_args.sh**

```
mlefranc@FAYOL-LEFRANCOIS:
  GNU nano 4.8            ./count_args.sh
echo $# arguments
echo program: $0
echo arg1: $1
echo arg2: $2
echo arg3: $3
echo arg4: $4
echo arg5: $5
echo arg6: $6
echo arg7: $7
echo arg8: $8
echo arg9: $9

                    [ Read 11 lines ]
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text
^X Exit       ^R Read File  ^\ Replace    ^U Paste Text
```

Run with a list of arguments

```
mlefranc@FAYOL-LEFRANCOIS:
mlefranc@FAYOL-LEFRANCOIS:/mnt/c/Users/maxime.lefrancois$ .
/count_args.sh commit --interactive -m "Corrected typo in h
1" --author="John Smith" index.html
6 arguments
program: ./count_args.sh
arg1: commit
arg2: --interactive
arg3: -m
arg4: Corrected typo in h1
arg5: --author=John Smith
arg6: index.html
arg7:
arg8:
arg9:
```

Run a program in the console

Name of the program, then list of arguments separated by spaces

example: small program **count_args.sh**

```
mlefranc@FAYOL-LEFRANCOIS:
  GNU nano 4.8            ./count_args.sh
echo $# arguments
echo program: $0
echo arg1: $1
echo arg2: $2
echo arg3: $3
echo arg4: $4
echo arg5: $5
echo arg6: $6
echo arg7: $7
echo arg8: $8
echo arg9: $9

                    [ Read 11 lines ]
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text
^X Exit       ^R Read File  ^\ Replace    ^U Paste Text
```

Use of the '\' line continuation symbol for clarity

```
mlefranc@FAYOL-LEFRANCOIS:
mlefranc@FAYOL-LEFRANCOIS:/mnt/c/Users/maxime.lefrancois$ ./count_args.sh \
>   commit --interactive \
>   -m "Corrected typo in h1" \
>   --author="John Smith" \
>   index.html
6 arguments
program: ./count_args.sh
arg1: commit
arg2: --interactive
arg3: -m
arg4: Corrected typo in h1
arg5: --author=John Smith
arg6: index.html
arg7:
arg8:
arg9:
```

Run a program in the console

Name of the program, then list of arguments separated by spaces

Single quote vs. double quote: compare

```
mlefranc@FAYOL-LEFRANCOIS:/mnt/c/Users/maxime.lefrancois$ ./count_args.sh \
>   commit --interactive \
>   -m 'Corrected typo in h1 at $(date)' \
>   --author="John Smith" \
>   index.html
6 arguments
program: ./count_args.sh
arg1: commit
arg2: --interactive
arg3: -m
arg4: Corrected typo in h1 at $(date)
arg5: --author=John Smith
arg6: index.html
arg7:
arg8:
arg9:
```

```
mlefranc@FAYOL-LEFRANCOIS:/mnt/c/Users/maxime.lefrancois$ ./count_args.sh \
>   commit --interactive \
>   -m "Corrected typo in h1 at $(date)" \
>   --author="John Smith" \
>   index.html
6 arguments
program: ./count_args.sh
arg1: commit
arg2: --interactive
arg3: -m
arg4: Corrected typo in h1 at Thu Aug 26 15:30:02 CEST 2021
arg5: --author=John Smith
arg6: index.html
arg7:
arg8:
arg9:
```

Run a program in the console

Name of the program, then list of arguments separated by spaces

example: small program **count_args.sh**

```
mlefranc@FAYOL-LEFRANCOIS:
  GNU nano 4.8            ./count_args.sh
echo $# arguments
echo program: $0
echo arg1: $1
echo arg2: $2
echo arg3: $3
echo arg4: $4
echo arg5: $5
echo arg6: $6
echo arg7: $7
echo arg8: $8
echo arg9: $9

                        [ Read 11 lines ]
^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text
^X Exit       ^R Read File  ^\ Replace    ^U Paste Text
```

Escape the space to avoid using quotation marks

```
mlefranc@FAYOL-LEFRANCOIS:
mlefranc@FAYOL-LEFRANCOIS:/mnt/c/Users/maxime.lefrancois$ ./count_args.sh \
>   commit --interactive \
>   -m Corrected\ typo\ in\ h1 \
>   --author=John\ Smith \
>   index.html
6 arguments
program: ./count_args.sh
arg1: commit
arg2: --interactive
arg3: -m
arg4: Corrected typo in h1
arg5: --author=John Smith
arg6: index.html
arg7:
arg8:
arg9:
```

Program CLI (command line interface)

With nearly 40 years of CLI design, some good conventions have been established
Read for example: **Command Line Interface Guidelines** https://clig.dev/

```
Naval Fate.

Usage:
  naval_fate ship new <name>...
  naval_fate ship <name> move <x> <y> [--speed=<kn>]
  naval_fate ship shoot <x> <y>
  naval_fate mine (set|remove) <x> <y> [--moored|--drifting]
  naval_fate -h | --help
  naval_fate --version

Options:
  -h --help     Show this screen.
  --version     Show version.
  --speed=<kn>  Speed in knots [default: 10].
  --moored      Moored (anchored) mine.
  --drifting    Drifting mine.
```

example of CLI documentation. Source, http://docopt.org/

Program CLI (command line interface)

By convention, each program (e.g. **git**) has a documentation, which can be obtained with the argument **help**, **-h** , or **--help**

The **man** command can also be used:

```
$ man git
```

to exit the documentation, press **q**

mlefranc@FAYOL-LEFRANCOIS:

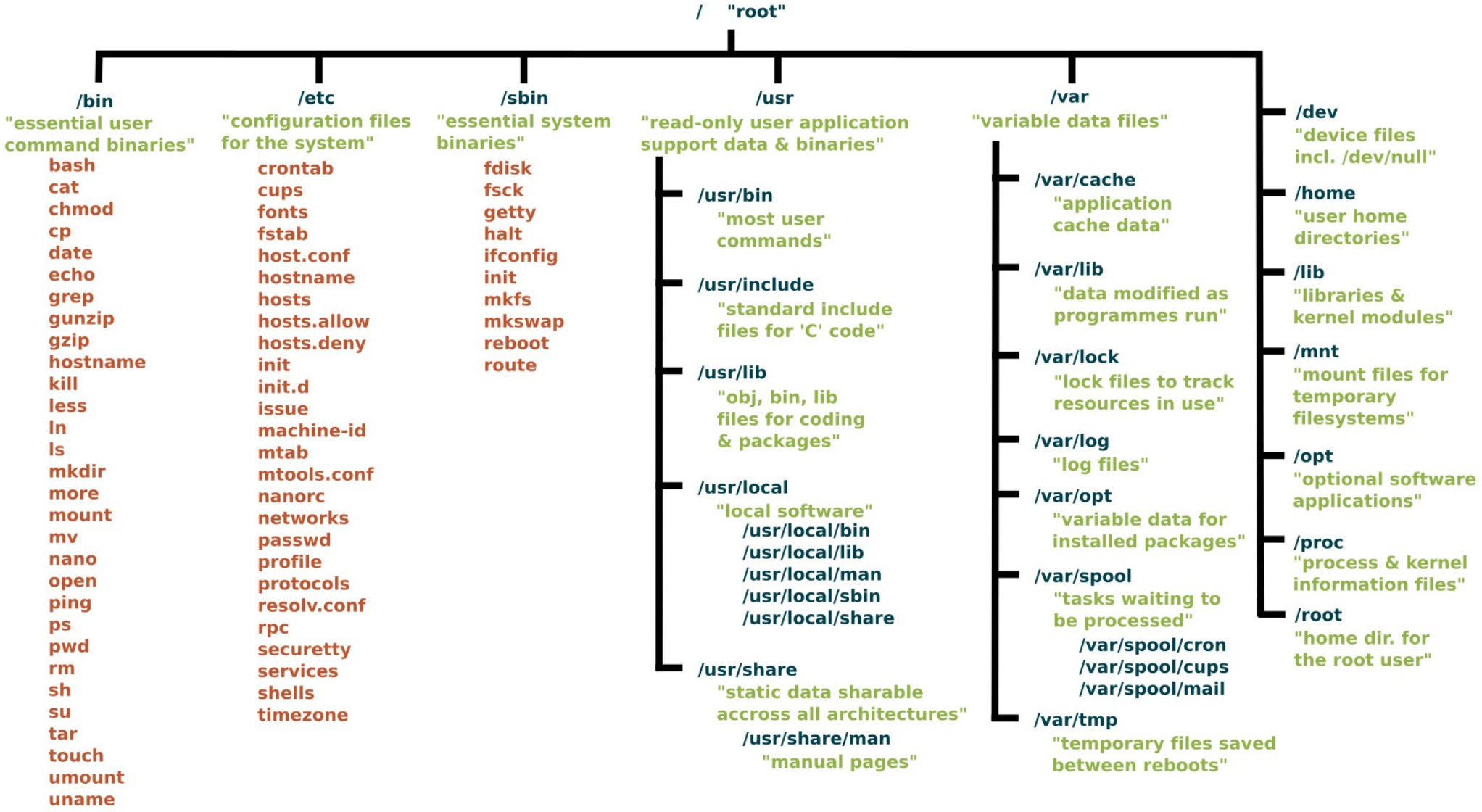
```
mlefranc@FAYOL-LEFRANCOIS:/mnt/c/Users/maxime.lefrancois$ git help
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone             Clone a repository into a new directory
   init              Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add               Add file contents to the index
   mv                Move or rename a file, a directory, or a symlink
   restore           Restore working tree files
   rm                Remove files from the working tree and from the index
   sparse-checkout   Initialize and modify the sparse-checkout

examine the history and state (see also: git help revisions)
   bisect            Use binary search to find the commit that introduced a bug
   diff              Show changes between commits, commit and working tree, etc
   grep              Print lines matching a pattern
   log               Show commit logs
   show              Show various types of objects
   status            Show the working tree status

grow, mark and tweak your common history
   branch            List, create, or delete branches
   commit            Record changes to the repository
```

Fundamental Linux principles

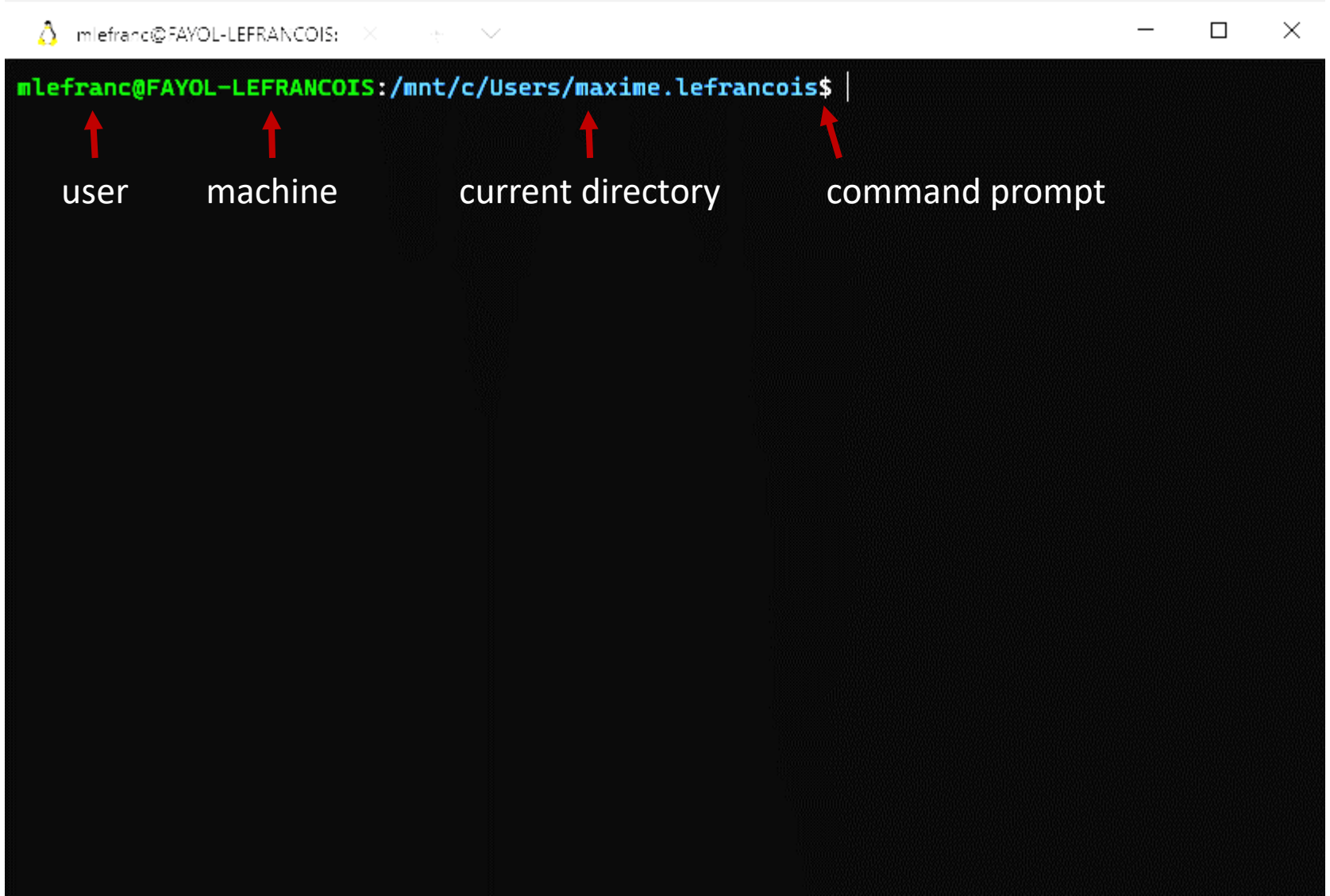
- Everything is a file. (Including hardware)
- Small, single-purpose programs.
- Ability to chain programs together to perform complex tasks.
- Avoid captive user interfaces.
- Configuration data stored in text.

Example justification: https://linuxtutorials4u.wordpress.com/2011/09/03/linux-principles/

Standard linux file system hierarchy



The command prompt



Navigating the file system

```
mlefranc@FAYOL-LEFRANCOIS:/mnt/c/Users/maxime.lefrancois$ cd ~
mlefranc@FAYOL-LEFRANCOIS:~$ pwd
/home/mlefranc
mlefranc@FAYOL-LEFRANCOIS:~$ ls
Desktop    Music     README.md  apache-jena-4.1.0      sshagent.sh
Documents  Pictures  Templates  apache-jena-4.1.0.zip  temp
Downloads  Public    Videos     maxime.lefrancois
mlefranc@FAYOL-LEFRANCOIS:~$ cd ..
mlefranc@FAYOL-LEFRANCOIS:/home$ ls
mlefranc
mlefranc@FAYOL-LEFRANCOIS:/home$ pwd
/home
mlefranc@FAYOL-LEFRANCOIS:/home$ cd ../mnt/c/Users
mlefranc@FAYOL-LEFRANCOIS:/mnt/c/Users$ /bin/echo hello
hello
mlefranc@FAYOL-LEFRANCOIS:/mnt/c/Users$ ../../../bin/echo hello
hello
mlefranc@FAYOL-LEFRANCOIS:/mnt/c/Users$ cd ./maxime.lefrancois/
mlefranc@FAYOL-LEFRANCOIS:/mnt/c/Users/maxime.lefrancois$ ls /
bin   dev  home  lib    lib64   lost+found  mnt  proc  run   snap  sys  usr
boot  etc  init  lib32  libx32  media       opt  root  sbin  srv   tmp  var
```

The permissions system

```
mlefranc@ci:~$ ls -al /var/www/html
total 80
drwxr-xr-x 18 root        root        4096 mars  11 17:17 .
drwxr-xr-x 19 root        root        4096 juil.  5 13:25 ..
lrwxrwxrwx  1 root        root          29 févr.  3  2020 ai4industry -> /home/ai4industr
y/ai4industry
drwxr-xr-x  3 gmartins    gmartins    4096 sept. 28  2020 apps
drwxr-xr-x 11 ai4industry ai4industry 4096 juin  22 15:34 cps2
lrwxrwxrwx  1 root        root          29 févr. 19  2021 data -> /home/coswot-data/coswo
t-data
drwxr-xr-x  5 mlefranc    d-bdata     4096 août  24 12:13 d-bdata
drwxr-xr-x  2 root        root        4096 nov.  30  2020 d-ia
lrwxrwxrwx  1 root        root           4 nov.  27  2020 d-tnrg -> d-ia
```

permissions

owner user

owner group

size (bytes)

last modif.

name -> link target

deuxième colonne: https://bertvv.github.io/notes-to-self/2015/10/18/the-number-of-hard-links-in-ls--l/

Julia Evans
@b0rk

unix permissions

drawings.jvns.ca

There are 3 things you can do to a file

read write execute

ls -l file.txt shows you permissions
Here's how to interpret the output:

rw-	rw-	r--	bork staff
bork (user) can read & write	staff (group) can read & write	ANYONE can read	

File permissions are 12 bits

setuid setgid

	user	group	all
000	110	110	100
sticky	rwx	rwx	rwx

For the r/w/x bits:

1 means "allowed"

0 means "not allowed"

110 in binary is 6

So	rw-	r--	r--
=	110	100	100
=	6	4	4

chmod 644 file.txt means change the permissions to:

rw- r-- r--

simple!

setuid affects executables

$ls -l /bin/ping

rws r-x r-x root root

this means ping always runs as root

setgid does 3 different unrelated things for executables, directories, and regular files

https://twitter.com/b0rk/status/982641594305273856

more about setuid and setgid: https://en.wikipedia.org/wiki/Setuid

Assign a variable

`fruit=banana` # assigns the value banana to the variable `fruit`

```
echo $fruit    # outputs « banana »
echo "$fruit"  # outputs « banana »
echo '$fruit'  # outputs « $fruit »
```

Parameter Expansion

see section " Parameter expansions ".
https://devhints.io/bash#parameter-expansions

```
name="John"
echo ${name}
echo ${name/J/j}    #=> "john" (substitution)
echo ${name:0:2}    #=> "Jo"   (slicing)
echo ${name::2}     #=> "Jo"   (slicing)
echo ${name::-1}    #=> "Joh"  (slicing)
echo ${name%hne}    #=> "Jo"   (remove suffix)
echo ${name^^}      #=> "JOHN" (all uppercase)
echo ${food:-Cake}  #=> $food or "Cake"  if food doesn't exist
```

Command return values

Linux commands return a numerical value.

`0` : ok. command **`true`** always returns 0
`≠0` : not ok. commande **`false`** always returns 1

`$?` : contains the return value of the previous command

`<command1> && <command2>` # <command2> executed only in case of success

`<command1> || <command2>` # <command2> executed only in case of failure

`<command1> ; <command2>` # <command2> always executed

Standard input and output of controls

Linux commands have by default 3 different file descriptors.

<u>Standard input (STDIN):</u> file descriptor &0,

Commands expect information from STDIN.

By default, triggers a keyboard input request. Otherwise, can read from a file

```
$ mail toto
>Hi
>How are you?
>^d (equivalent to CTRL+d)
$
```

```
$ mail < file
$ mail 0< file
```

executes mail with the file contents as standard input

idem explicit file descriptor number

https://quennec.fr/trucs-astuces/syst%C3%A8mes/gnulinux/programmation-shell-sous-gnulinux/les-m%C3%A9canismes-du-shell/redirections

Standard input and output of controls

Linux commands have by default 3 different file descriptors.

<u>Standard output (STDOUT):</u> file descriptor &1,

By default, displayed on the screen. Can be redirected to a file

```
$ ls > temp
$ ls 1> temp
$ ls >> temp
$ ls 1>> temp
```

redirects the result of **ls** to the **temp** file
idem
concatenates the result of **ls** to the **temp** file
idem

https://quennec.fr/trucs-astuces/syst%C3%A8mes/gnulinux/programmation-shell-sous-gnulinux/les-m%C3%A9canismes-du-shell/redirections

Standard input and output of controls

Linux commands have by default 3 different file descriptors.

<u>Standard error output (STDERR):</u> file descriptor &2,

By default, displayed on the screen. Can be redirected to a file

```
$ command 2> temp
$ command 2>> temp
```

redirects errors from **command** to the **temp** file
concatenates errors from **command** to the **temp** file

https://quennec.fr/trucs-astuces/syst%C3%A8mes/gnulinux/programmation-shell-sous-gnulinux/les-m%C3%A9canismes-du-shell/redirections

Standard input and output of controls

Linux commands have by default 3 different file descriptors.

You can combine the redirections

```
$ find /etc -name smb.conf 1> result 2> error
$ cat result
/etc/samba/smb.conf
$ cat error
find: "/etc/lvm/cache": Permission denied
find: "/etc/lvm/backup": Permission denied
find: "/etc/lvm/archive": Permission denied
```

One can redirect to **/dev/null**

```
$ find /etc -name smb.conf 1> result 2> /dev/null
$ cat result
/etc/samba/smb.conf
$ cat /dev/null
$
```


https://quennec.fr/trucs-astuces/syst%C3%A8mes/gnulinux/programmation-shell-sous-gnulinux/les-m%C3%A9canismes-du-shell/redirections

Communication pipes

- The communication pipe (character '|') forwards the standard output of the left command to the standard input of the right command

```
$ ls | wc -l
29

$ ls | wc -l | mail toto

$ cat /etc/passwd | grep root | cut -d':' -f1,7
root:/bin/bash
```

Display the number of files in a directory

Send by mail the number of files in a directory

Display only certain elements of a file

Globbing

Character '*' allows to replace any sequence of characters

```
$ ls
file1 file2.a myfile herfile.b
$ ls *.a
file2.a
$ ls f*
file1 file2.a
```

The '?' character represents any character

```
$ ls *.?
file2.a herfile.b
$ ls ?????
file1
```

The '[]' characters allow you to indicate a list of characters you are looking for

```
$ ls [fh]*.[a-z]
File2.a herfile.b
$ ls ?[A-Z0-9e]*
cOucou  f1chier  F2chier  Hello
$ ls  [!a-z]*
1coucou  Coucou  F2chier  Fichier  Hello
```


https://quennec.fr/trucs-astuces/syst%C3%A8mes/gnulinux/programmation-shell-sous-gnulinux/les-m%C3%A9canismes-du-shell/remplacement-de-noms-de-fichiers/expressions-simples

Control structures

see « Loops » section
https://devhints.io/bash#loops

Basic for loop

```
for i in /etc/rc.*; do
  echo $i
done
```

C-like for loop

```
for ((i = 0 ; i < 100 ; i++)); do
  echo $i
done
```

Ranges

```
for i in {1..5}; do
    echo "Welcome $i"
done
```

With step size

```
for i in {5..50..5}; do
    echo "Welcome $i"
done
```

Reading lines

```
cat file.txt | while read line; do
  echo $line
done
```

Forever

```
while true; do
  ...
done
```

Conditions

See « Conditionals » section
https://devhints.io/bash#conditionals

`[[ -z STRING ]]`	Empty string
`[[ -n STRING ]]`	Not empty string
`[[ STRING == STRING ]]`	Equal
`[[ STRING != STRING ]]`	Not Equal
`[[ NUM -eq NUM ]]`	Equal
`[[ NUM -ne NUM ]]`	Not equal
`[[ NUM -lt NUM ]]`	Less than
`[[ NUM -le NUM ]]`	Less than or equal
`[[ NUM -gt NUM ]]`	Greater than
`[[ NUM -ge NUM ]]`	Greater than or equal
`[[ STRING =~ STRING ]]`	Regexp
`(( NUM < NUM ))`	Numeric conditions
`[[ ! EXPR ]]`	Not
`[[ X && Y ]]`	And
`[[ X || Y ]]`	Or

`[[ -e FILE ]]`	Exists
`[[ -r FILE ]]`	Readable
`[[ -h FILE ]]`	Symlink
`[[ -d FILE ]]`	Directory
`[[ -w FILE ]]`	Writable
`[[ -s FILE ]]`	Size is > 0 bytes
`[[ -f FILE ]]`	File
`[[ -x FILE ]]`	Executable
`[[ FILE1 -nt FILE2 ]]`	1 is more recent than 2
`[[ FILE1 -ot FILE2 ]]`	2 is more recent than 1
`[[ FILE1 -ef FILE2 ]]`	Same files

```bash
# String
if [[ -z "$string" ]]; then
  echo "String is empty"
elif [[ -n "$string" ]]; then
  echo "String is not empty"
else
  echo "This never happens"
fi
```

```bash
# Equal
if [[ "$A" == "$B" ]]
```

```bash
# Regex
if [[ "A" =~ . ]]
```

```bash
if (( $a < $b )); then
   echo "$a is smaller than $b"
fi
```

```bash
if [[ -e "file.txt" ]]; then
  echo "file exists"
fi
```

TODO read: http://mywiki.wooledge.org/BashFAQ/031

Invoke a script

You can call a script
example executable file (see permissions) `./reverse`

```
for ((i=$# ; i>0 ; i--)); do
  echo ${!i};
done
```

```
$ reverse a b c
c
b
a
```

You can add a "shebang" line at the beginning of the file to specify the interpreter to use
example executable file (see permissions) `./reverse`

```
#!/bin/bash
for ((i=$# ; i>0 ; i--)); do
  echo ${!i};
done
```

```
$ reverse a b c
c
b
a
```

So you can use other interpreters, python, perl, etc.
example executable file (see permissions) `./reverse`

```
#!/usr/bin/python3
import sys
for arg in reversed(sys.argv[1:]):
    print(arg)
```

```
#!/usr/bin/env python3
import sys
for arg in reversed(sys.argv[1:]):
    print(arg)
```

```
$ ./reverse a b c
c
b
a
```

Top 50 unix commands (for this course)

Users and permissions

sudo Command to escalate privileges in Linux
useradd and usermod - Add new user or change existing users data
passwd Create or update passwords for existing users
chmod Command to change file permissions
chown Command for granting ownership of files or folders

System, terminal, processes

whoami Print effective userid
apt, pacman, yum, rpm - Package managers depending on the distro
date Print or set the system date and time
clear Clear the terminal display
exit Cause normal process termination
man Access manual pages for all Linux commands
df Report file system disk space usage
du Estimate file space usage
ps Report a snapshot of the current processes

File system

ls Command in Linux to list files
pwd Print working directory command in Linux
cd Linux command to navigate through directories
mkdir Command used to create directories in Linux
mv Move or rename files in Linux
cp Similar usage as mv but for copying files in Linux
rm Delete files or directories
touch Create blank/empty files
ln Create symbolic links (shortcuts) to other files
comm Combines the functionality of diff and cmp
find Search for files in a directory hierarchy

Reading and writing

read Read a file descriptor in variables
echo Print any text that follows the command
printf Command identical to the C language one

Top 50 unix commands (for this course)

Environment and position variables

env	Display environment variables
unset	Unset an environment variable
set	set/unset values of shell options and positional parameters
shift	Shift arguments to the left ($2 becomes $1)
export	Export environment variables in Linux

Filter commands – data visualisation

cat	Display file contents on the terminal
head	Return the specified number of lines from the top
grep	Search for a string within an output
sed	Stream editor for filtering and transforming text
tail	Return the specified number of lines from the bottom
tee	Read from STDIN and write to standard output and files
nl	Print file with line numbers

Filter commands – data processing

cut	Remove sections from each line of files
wc	Print newline, word, and byte counts for each file
sort	Sort lines of text files
paste	Merge lines of files
split	Split a file into pieces
tr	Translate or delete characters
uniq	Report or omit repeated lines

Compression, archiving, conversion

tar	Command to extract and compress files in Linux
zip	Zip files in Linux
unzip	Unzip files in Linux
dd	Majorly used for creating bootable USB sticks

Web

wget	Direct download files from the internet
curl	Transfer a URL

... some references to deepen the subject

@fr: parcourez les notes de Ronan Quennec:
https://quennec.fr/trucs-astuces/syst%C3%A8mes/gnulinux/programmation-shell-sous-gnulinux

Cheatsheet at devhints: https://devhints.io/bash
For the record, almost everything you see about Bash applies to zsh : https://devhints.io/zsh

Simplified man pages https://tldr.ostera.io/

… your turn

Complete the TODO section:

https://ci.mines-stetienne.fr/cps2/course/tfsd/course-1.html#_todos