# Maitriser l’environnement Raspberry pour la mise en place d’une solution IoT

Objectifs :

# Partie 01: Prise en main du Raspberry Pi

# Partie 01 : Préparation de l’environnement

Dans cette partie on va préparer l’environnement de travail composé de :

* Raspberry PI Board
* SD card and Reader
* OS Rasbian

# Logiciels nécessaires

<https://www.raspberrypi.org/downloads/>

* Un fichier Raspbian.iso (.img)
* BalenaEtcher ou (SDFormatter et Win32DiskImager)
* MobaXterm ou Putty pour accès à distance

# Préparation de la carte MicroSD:

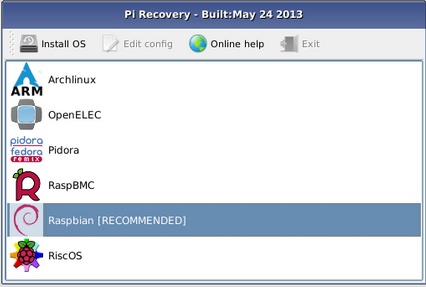
* Formater la carte SD avec SDFormatter sinon si vous allez utiliser BalenaEtcher, ignorer cette étape.

# Installation du système Linux sous Raspberry

Deux méthodes d'installation sont possibles :

## Méthode1: Utilisation de l'installateur Noobs (méthode qui prend du temps et n’est pas recommandée)

1. Télécharger le fichier .zip de **NOOBS** (si la version a changé, le lien risque d'être rompu, rendez-vous alors sur la page de téléchargement)
2. le décompresser.
3. copier tous les fichiers obtenus sur la carte SD.
4. Insérer la carte SD dans le **Raspberry** Pi.
5. allumer le **Raspberry** Pi.



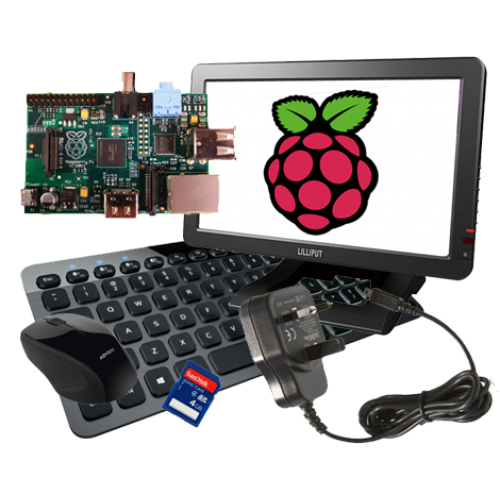
## Méthode2: Utilisation de l'image système Raspbian (recommandée)

1. Graver l'image système Raspbian.iso (.img) sur la carte SD : utiliser le logiciel BalenaEtcher (recommandé)
2. Insérer la carte mémoire et alimenter la carte RaspberryPi.

# Partie 02 : Connexion à la carte Raspberry

## Clavier souris et écran

Utiliser un clavier, une souris et un câble HDMI avec un écran et amuser vous avec votre micro-ordinateur.



## Sinon un accès à distance est obligatoire.

## SSH remote access

* **Insérez votre carte microSD de nouveau dans votre PC**
* Créez un fichier sans extension de nom « ssh » sous le répertoire **/boot** dans la carte microSD

Définir un réseau sans fil (WiFi) :

Afin de permettre la connexion de la carte Raspberry à un réseau WiFi, une petite opération doit être faite au niveau du dossier « /boot » dans la carte microSD.

Créer un fichier dont le nom est « wpa\_supplicant.conf » sous le dossier « /boot » dans lequel il faut inscrire :

ctrl\_interface=DIR=/var/run/wpa\_supplicant GROUP=netdev

update\_config=1

country=TN

network={

ssid="<Name of your wireless LAN>"

psk="<Password for your wireless LAN>"

}

Une fois cette opération est finie, remettre la carte microSD dans la carte Raspberry et alimentez la carte.

Pour détecter l’adresse de la carte Raspberry, il faut regarder les hôtes connectés sur le réseau et localiser le hôte « Raspberry » dont l’adresse IP va permettre de se connecter à distance sur la carte. Ou utiliser les utilitaires convenables pour scanner votre réseau :

* Ping raspberrypi
* En uilisant Wireshark et DHCP
* En utilisant nmap
* En utilisant netscan

## Accès console :

1. **Hôte Windows :**

Depuis un hote Windows, on peut se connecter à distance sur la carte Raspberry en utilisant les utilitaire disponibles suivants :

* MobaXterm
* Putty

1. **Hôte linux**
   * 1. Depuis un hôte linux connecté au réseau, ouvrez une session SSH avec la Raspberry PI dans un terminal. Explicitez les différentes étapes.

**ssh <IP\_Raspberry\_PI>**

* + 1. La commande **scp** permet de copier un fichier ou un répertoire (**-r**) d’un client vers un serveur ou d’un serveur vers un client. Depuis votre hôte linux, copiez le fichier **hello1.cpp** présent sur la Raspberry PI dans votre dossier personnel.

**scp <Fichier\_local> <login@IP\_client\_distant:Chemin>**

**scp <login @IP\_serveur\_distant:Chemin/Fichier> <Chemin/Fichier\_local>**

Remarque: login: pi /mot de passe raspberry

## Bureau à distance

* La commande mstsc coté client en indiquant l’adresse IP
* Configuration coté carte Raspberry :
  + sudo apt-get install xrdp
  + sudo apt-get install tightvncserver

Une fois nous sommes connectés à la carte, nous devons passer à la :

# Mise à jour de l’environnement

Commandes nécessaires après installation :

* sudo apt-get update
* sudo apt-get upgrade

**Compiling your first program “LED Blink”**

Le Raspberry Pi offre quelques possibilités d’entrées-sorties directes en utilisant les broches GPIO présentes sur son connecteur. Elles ne sont pas très nombreuses (une dizaine) mais cela peut suffire pour des petits projets interactifs nécessitant d’interroger des capteurs tout-ou-rien ou de valider des actionneurs.

Nous pouvons utiliser ces GPIO de différentes façons, depuis l’espace utilisateur ou depuis le noyau. Le but ici est de balayer les principaux aspects.



1. **L’interface sysfs**

Sysfs est un système de fichiers virtuel introduit par le noyau Linux 2.6. Sysfs permet d'exporter depuis l'espace noyau vers l'espace utilisateur des informations sur les périphériques du système et leurs pilotes, et est également utilisé pour configurer certaines fonctionnalités du noyau.

Pour chaque objet ajouté dans l'arbre des modèles de pilote (pilotes, périphériques, classes de périphériques), un répertoire est créé dans sysfs.

* + 1. La relation parent/enfant est représentée sous la forme de sous-répertoires dans **/sys/devices/** (représentant la couche physique).
    2. Le sous-répertoire **/sys/bus/** est peuplé de liens symboliques, représentant la manière dont chaque périphérique appartient aux différents bus.
    3. **/sys/class/** montre les périphériques regroupés en classes, comme les périphériques réseau par exemple.

Pour les périphériques et leurs pilotes, des attributs peuvent être créés. Ce sont de simples fichiers, la seule contrainte est qu'ils ne peuvent contenir chacun qu'une seule valeur et/ou n'autoriser le renseignement que d'une valeur. Ces fichiers sont placés dans le sous-répertoire du pilote correspondant au périphérique.

1. **Lire/Ecrire sur une broche TOR**

**Accès depuis l’espace utilisateur**

L’accès simple, depuis le shell peut se faire très aisément grâce au système de fichiers **/sys**.

/ # **cd /sys/class/gpio/**

/sys/class/gpio # **ls**

export gpiochip0 unexport

Accédons au GPIO 24 (broche 18) :

/sys/class/gpio # **echo 24 > export**

/sys/class/gpio # **ls**

export gpio24 gpiochip0 unexport

/sys/class/gpio # **cd gpio24/**

/sys/class/gpio/gpio24 # **ls**

active\_low direction edge subsystem uevent value

**Ecriture**

La broche doit être configurée en sortie pour écrire :

/sys/class/gpio/gpio24 # **echo out > direction**

/sys/class/gpio/gpio24 # **echo 1 > value**

/sys/class/gpio/gpio24 # **echo 0 > value**

**Lecture**

La broche doit être configurée en entrée pour lire :

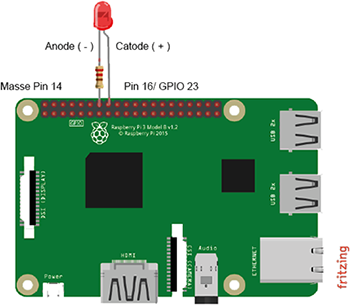
/sys/class/gpio/gpio24 # **echo in > direction**

/sys/class/gpio/gpio24 # **cat value**

0

**Programmation des GPIO**

L’utilisation des ports GPIO est liée à l’accès à plusieurs fichiers. Les opérations d’export, de configuration de la direction, de lecture ou d’écriture peuvent être réalisées dans n’importe quel langage de programmation.



**Programme 1 : Clignoter une LED en C**

1. Câbler le circuit
2. Copier le code suivant en utilisant la commande
   1. Sudo nano led.c

#include <stdio.h>

#include <wiringPi.h>

// LED Pin - wiringPi pin 0 is BCM\_GPIO 17 n°Pin 11.

#define LED 4

int main (void)

{

printf ("Raspberry Pi blink\n") ;

wiringPiSetup () ;

pinMode (LED, OUTPUT) ;

for (;;)

{

digitalWrite (LED, HIGH) ; // On

delay (500) ; // mS

digitalWrite (LED, LOW) ; // Off

delay (500) ;

}

return 0 ;

}

1. Installer la bibliothèque :
   1. cd /tmp
   2. wget <https://project-downloads.drogon.net/wiringpi-latest.deb>
   3. sudo dpkg -i wiringpi-latest.deb
2. Compilez le code : gcc led.c -o led -lwiringPi
3. Exécutez : ./led

**Programme 02 : Python**

$ sudo apt-get install python-rpi.gpio python3-rpi.gpio

import RPi.GPIO as GPIO # Import Raspberry Pi GPIO library

from time import sleep # Import the sleep function from the time module

GPIO.setwarnings(False) # Ignore warning for now

GPIO.setmode(GPIO.BOARD) # Use physical pin numbering

GPIO.setup(8, GPIO.OUT, initial=GPIO.LOW) # Set pin 8 to be an output pin and set initial value to low (off)

while True: # Run forever

GPIO.output(8, GPIO.HIGH) # Turn on

sleep(1) # Sleep for 1 second

GPIO.output(8, GPIO.LOW) # Turn off

sleep(1) # Sleep for 1 second

$ python blinking\_led.py

**Programme 03 : shell**

#!/bin/bash

GPIO=23

GPIODIR=/sys/class/gpio/gpio$GPIO

echo "Configuring GPIO $GPIO"

#check if the gpio is already exported

if [ ! -e "$GPIODIR" ]

then

echo "Exporting GPIO"

echo $GPIO > /sys/class/gpio/export

else

echo "GPIO already exported"

fi

echo "Current direction: `cat $GPIODIR/direction`"

echo "Set GPIO as output"

echo out > $GPIODIR/direction

echo "New GPIO direction: `cat $GPIODIR/direction`"

echo "Current value: `cat $GPIODIR/value`"

echo "Set value as high"

echo 1 > $GPIODIR/value

echo "New value: `cat $GPIODIR/value`"

#Endless loop

echo "Start blinking, 1 sec on plus 1 sec off, press CTRL+C to end"

while ( true );

do echo 1 > $GPIODIR/value;

cat $GPIODIR/value;

sleep 1;

echo 0 > $GPIODIR/value;

cat $GPIODIR/value;

sleep 1;

done;

**Programme 04 : C++**

#include <iostream>

#include <fstream>

#include <unistd.h>

#include <string>

// Remplacez les ... par le code approprié

#define LED24 "/sys/class/gpio/ ..."

#define BUTTON "/sys/class/gpio/ ... "

using namespace std;

int main(...)

{

ofstream led24; ifstream button;

while(1)

{

button.open(...);

string buttonState;

getline(button,buttonState);

button23.close();

if(buttonState=="1")

{

led24.open(LED24);

//Allumer led24 << "...";

led24.close();

sleep(...);

led24.open(LED24);

//Eteindre led24 << "...";

led24.close();

sleep(...);

}

}

}