

**Compte rendu   
  
Belhadj Walid   
uapv2101413**

# Raspberry pi - le matériel

<https://en.wikipedia.org/wiki/Raspberry_Pi>

* **Quel est la référence du processeur qui équipe votre carte Raspberry Pi 2 B?**

La référence du processeur de la carte Raspberry PI 2 B est : **ARM Cortex-A7**

* **Quel est son type d’architecture ? Combien de cœurs possède-t-il ?**

Type d’architecture : 32 bits, et quad-core (4 cœurs).

* **Quel est la quantité de RAM disponible ?**

Ram de 1 Go.

* **La mémoire est-elle gérée par une MMU ?**

Oui.

* **Quel est le ratio entre la mémoire CPU/GPU par défaut ?**

GPU 250 Mhz, CPU : 900 MHz, ratio : 900/250= 3.6

* **Quels sont les bus de communication disponibles ?**

Les bus disponibles sont ; I2C, PICAN-2 CAN-bus, ( controller area network), SPI RPI, GPIO

* **Quelle version de VFP (Vector Foating Point) support-il ?**

Version 4

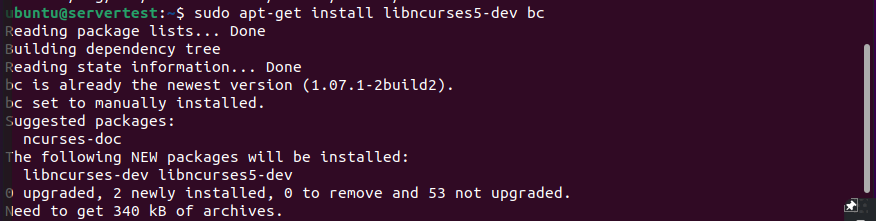
# Buildroot pour Raspberry Pi 2 et QEMU

## Installation de l’environnement Buildroot

**Buildroot** est un outil en ligne de commande permettant de générer sa propre image de GNU/Linux optimisée pour l'embarqué.

Commencez par installer les dépendances nécessaires à l'utilisation de Buildroot :

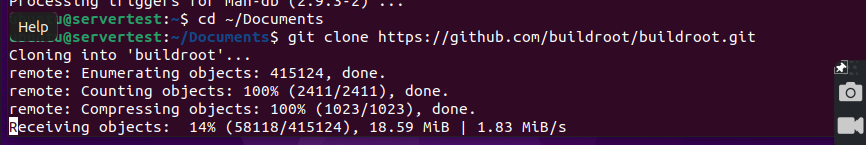
sudo apt-get install libncurses5-dev bc



Clonez maintenant les sources de l'outil **Buildroot** dans votre dossier de personnel :

cd ~/Documents

git clone <https://github.com/buildroot/buildroot.git>

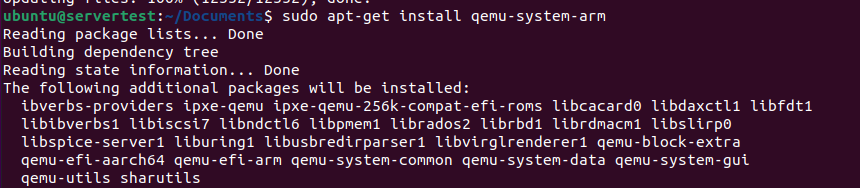


## Construction d’un système de base

Vous pouvez maintenant générer votre première image minimale de GNU/Linux à partir de **Buildroot**.

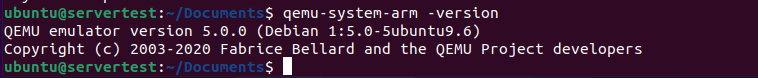
Comme nous ne disposons pas d’une carte Raspberry, nous allons utilisé QEMU pour l’émuler

## Installation de QEMU



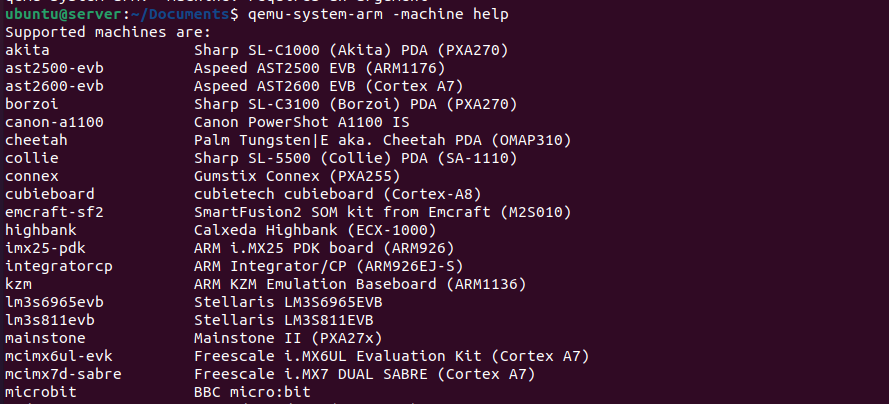
sudo apt-get install qemu-system-arm

* Assurez vous que la version de qemu dont vous disposez est bien supérieure à 2.4



Qemu-system-arm -version

* Listez l'ensemble des plateformes (machines) supportées par QEMU :



* Assurez vous de la prise charge de la plateforme Raspberry pi 2



qemu-system-arm -machine help

* Listez les processeurs supportés par l'émulateur pour cette plateforme.

qemu-system-arm -machine raspi2 -cpu help

* Assurez vous que le processeurs qui équipe le Raspberry pi 2B est bien supporté par l’émulateur



## Génération de la première image

* + - * Prenez connaissance du fichier **README**, quelles sont les actions à mener pour construire un système ?

Déjà, il faudra être en super utilisateur pour construire un système et suivre ces instructions suivantes :   
1) On lance la commande **'make menuconfig'**

2) On sélectionne l’architecture cible et les packages qu’on souhaite compiler.

3) On lance la commande **'make'**

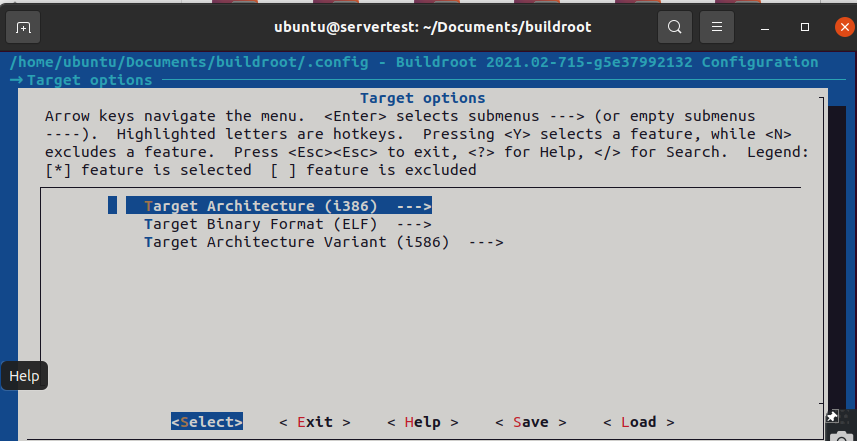
4) On patiente un peu la fin de la construction .

5) On cherche le noyau, le bootloader, le filesystem racine, etc. dans output/images

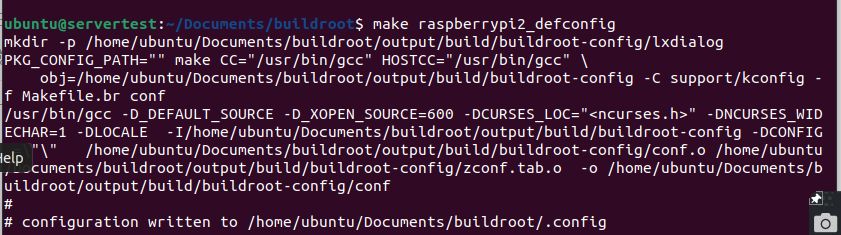
D’une manière plus large on peut définir les étapes de construction d’un système comme suit :   
Installer des bibliothèques de développement

1. Préparer le dossier cible
2. Créer un dossier de travail et définir une chaîne d'outils
3. Créer et configurer une racine système
4. Télécharger Qt
5. Configurer Qt pour la compilation croisée
6. Compiler, installer et déployer Qt
7. Configurer Qt Creator pour la compilation croisée Raspberry Pi
   * + - Consultez le menu de configuration de buildroot par défaut, notamment **Target options**.

~/Documents/buildroot$ make menuconfig



* + - * Buildroot contient des fichiers de pré-configuration pour un grand nombre de plateformes. Prenez connaissance du fichier de définition de la configuration pour une cible Raspberry Pi 2. Les fichiers de configuration standard des différentes cibles supportées par Buildroot se trouvent dans le dossier **configs**. matériel.
      * Appliquez le fichier de configuration standard pour le Raspberry Pi 2 :



~/Documents/buildroot$ make raspberrypi2\_defconfig

* + - * Observez la création du fichier **.config**.
      * Consultez le fichier **readme.txt** du dossier **board/raspberrypi2**.
      * Les différentes étapes à mener pour compiler :

La toute première étape est de créer une configuration.

1. On configure **Buildroot**: pour notre cas raspberry pi model 2 B « **make raspberrypi2\_defconfig** »
2. on construit le file system racine ( rootfs) « **make ».**

La configuration make ( make, make menuconfig ..etc ) permet de:

• onfigurer, construire et installer la chaîne d'outils de compilation croisée, ou simplement importer une chaîne d'outils externe;

• configurer, construire et installer les packages cibles sélectionnés;

• construire une image du noyau, si elle est sélectionnée;

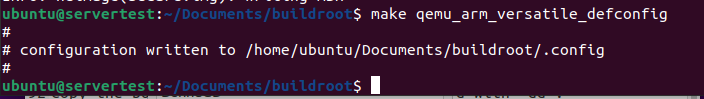
• créer une image du chargeur de démarrage, si elle est sélectionnée;

• créer un système de fichiers racine dans des formats sélectionnés

1. Il reste maintenant de déployer le système sur une carte micro-SD

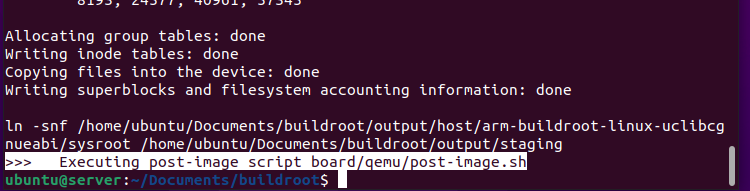
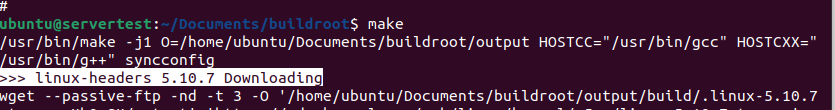
BuildRoot est un projet en développement rapide et de nouvelles fonctionnalités sont souvent ajoutées.  
BuildRoot a une interface de configuration similaire à l'interface de configuration du noyau Linux. Pour lancer l'interface de configuration

~/Documents/buildroot$ make



* + - * Lancez la compilation du système (vous devez être connecter à internet) :

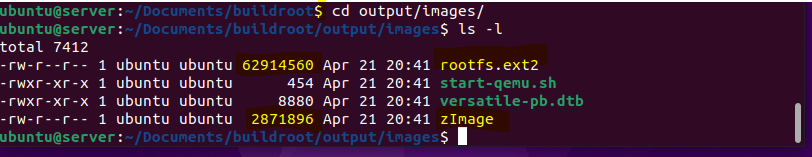
~/Documents/buildroot$ make



* + - * À l'issue de la génération, vous obtenez 2 fichiers dans le dossier output/images/. Le noyau correspond au fichier **zImage** et le système racine au fichier **rootfs.ext2**

~/Documents/buildroot$ cd output/images/

~/Documents/buildroot$ ls -l



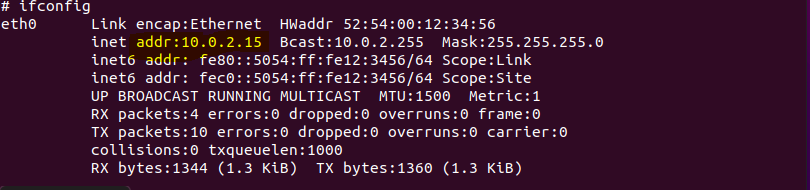
* + - * Quelle est la taille du noyau et du système de fichier (en Mo).

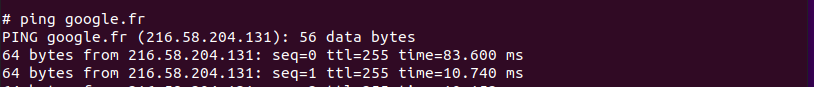
**62.91456 Mo pour rootfs.ext2 et 2.871896 Mo pour zImage**

* + - * Démarrez le système dans **QEMU** à l’aide du script généré par **Buildroot** :

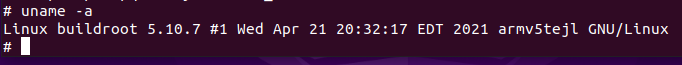
~/Documents/buildroot/output/images$ ./start-qemu.sh

* + - * Testez votre système et commentez les éléments suivant:
* Estimez la durée de boot : **15 seconds**
* Bannière d’accueil, mot de passe root
* Allure du prompt, disposition du clavier, **#** (**AZERTY** )
* Connexion réseau, connexion à distance par ssh

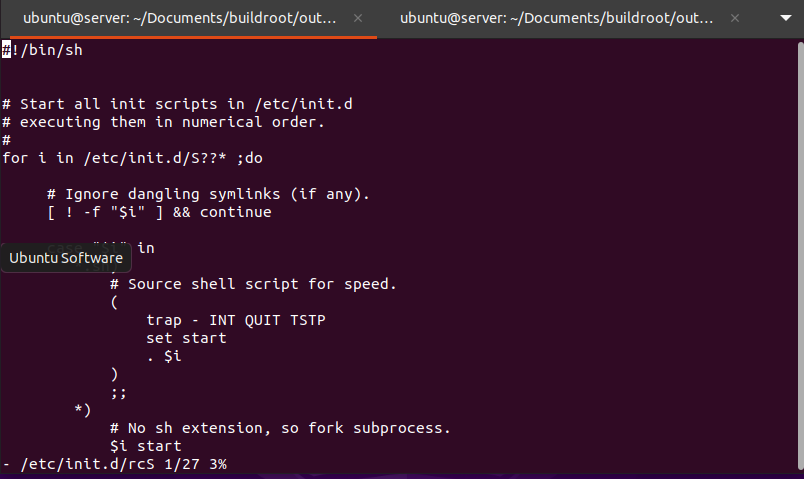




* Identifiez la version du noyau linux installé.



* + - * Consultez le script /etc/init.d/rcS



**Processus de démarrage du système :**

Lorsqu'un ordinateur x86 est démarré, le processeur regarde la fin de la mémoire système pour le BIOS (Basic Input / Output System) et l'exécute. Le programme BIOS est écrit dans une mémoire permanente en lecture seule et est toujours disponible pour utilisation. Le BIOS fournit l'interface de niveau le plus bas aux périphériques et contrôle la première étape du processus de démarrage.

Le BIOS teste le système, recherche et vérifie les périphériques, puis recherche un lecteur à utiliser pour démarrer le système. Habituellement, il vérifie le lecteur de disquette (ou le lecteur de CD-ROM sur de nombreux systèmes plus récents) pour un support de démarrage, le cas échéant, puis il regarde le disque dur. L'ordre des lecteurs utilisés pour le démarrage est généralement contrôlé par un paramètre BIOS particulier du système. Une fois Linux installé sur le disque dur d'un système, le BIOS recherche un Master Boot Record (MBR) à partir du premier secteur du premier disque dur, charge son contenu en mémoire, puis lui transmet le contrôle.

Ce MBR contient des instructions sur la façon de charger le chargeur de démarrage GRUB (ou LILO), en utilisant un système d'exploitation présélectionné. Le MBR charge ensuite le chargeur de démarrage, qui prend en charge le processus (si le chargeur de démarrage est installé dans le MBR). Dans la configuration par défaut de Red Hat Linux, GRUB utilise les paramètres du MBR pour afficher les options de démarrage dans un menu. Une fois que GRUB a reçu les instructions correctes pour que le système d'exploitation démarre, que ce soit à partir de sa ligne de commande ou du fichier de configuration, il trouve le fichier de démarrage nécessaire et confie le contrôle de la machine à ce système d'exploitation.

## Amélioration du système

### Prompt du shell global

Le système minimal présente un shell minimaliste dans lequel il n’est pas aisé de savoir où l’on se trouve dans le système de fichier et où l’utilisateur connecté n’est pas identifié clairement. On sait juste s’il s’agit ou non de root (**$** ou **#**).

Pour configurer le prompt du shell afin qu’il indique clairement l’utilisateur connecté et le dossier courant, il faut modifier le fichier **/etc/profile**. Le prompt du shell d’un terminal y est représenté par la variable **PSn** ou n est le numéro du terminal. Ici, nous n’avons qu’un seul terminal configuré : **PS1**

Pour obtenir un prompt de type **user@hostname : directory**

if [ "$PS1" ]; then

**export PS1="\u@\h:\w "**

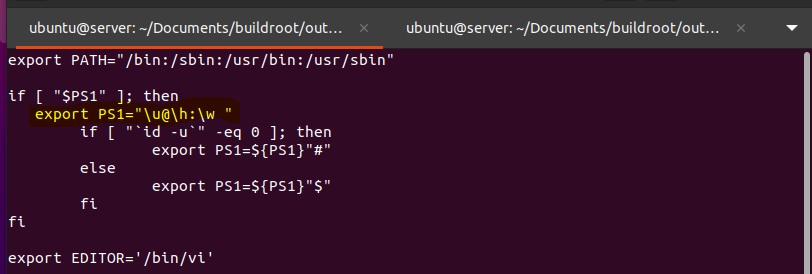
if [ "`id -u`" -eq 0 ]; then export PS1=**${PS1}**"# "

else

export PS1=**${PS1}**"$ "

fi

* + - * Modifiez le fichier **/etc/profile** comme ci-dessus.



* + - * Redémarrer la cible et observez l’allure du prompt quand vous changez de dossier.



* + - * Expliquez le test **if [ "`id -u`" -eq 0 ]; then** ... du script.

Cette ligne de code est une exécution de commande **« id -u »** test de l’ ID user s’il est à 0 (init) ça veut dire utilisateur principal ou **root** donc on exécute l’instruction export PS1=**${PS1}**"# " qui nous permet d’avoir le prompt **# ( root )**

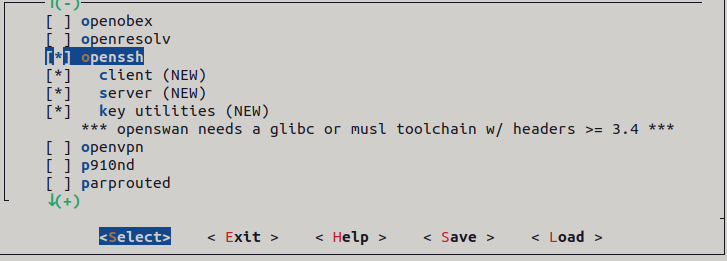
**Sinon** on exécute l’instruction export PS1=**${PS1}**"$" un utilisateur normal.

### Accès à distance

La prise en main à distance d'un système est en général assuré au moyen d’un serveur ssh. Vous pourrez utiliser au choix le package dropbear ou openSSH. Nous utiliserons ici **openSSH** :

### make menuconfig → Target packages → Networking applications

**→ openssh**

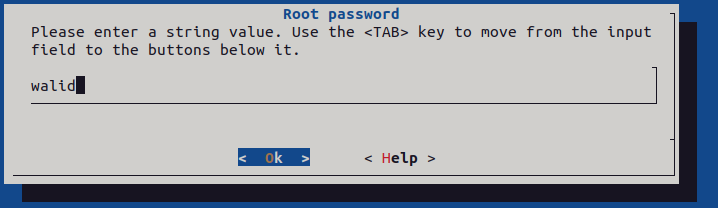


Dans les deux cas, il faut au préalable sécuriser l’accès local au système en créant un mot de passe pour root et un au moins un utilisateur local.

* + - * Sécurisez l'accès au système en attribuant un mot de passe à root :

### make menuconfig -> System configuration -> Root password

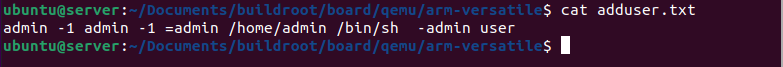
**Assurez vous de bien noter dans votre rapport de TP le mot de passe que vous avez attribué à root**



La création de comptes d'utilisateurs locaux se fait au moyen d'un fichier

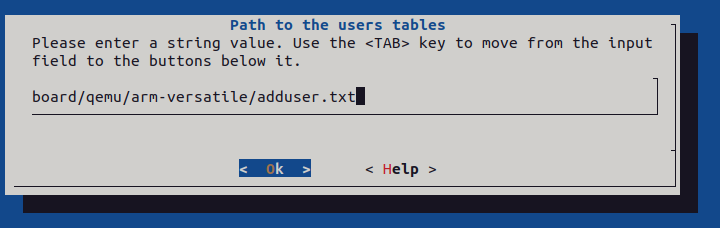
texte qui répond à la syntaxe de Makeusers définie dans le **chapitre 24** de la documentation de Buildroot :

<https://buildroot.org/downloads/manual/mancdual.html>



Indiquez à Buildroot la précd sence de ce fichier pour créer notre utilisateur conformément au paragraphe **9.6. Adding custom user accounts** de la documentation de Buildroot :

### make menuconfig → System configuration → Path to the users tables

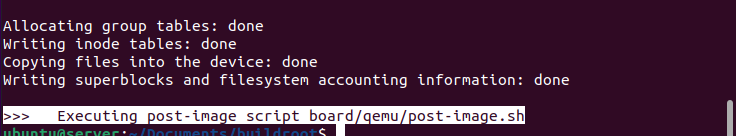


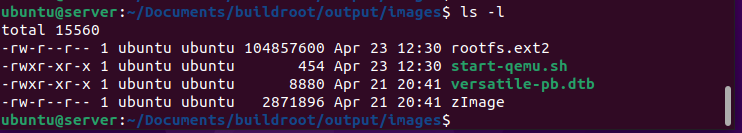
* + - * Modifiez la taille exact du système de fichier à 100Mo

### make menuconfig → Filesystem Image → exact size

### 

* + - * Relancez le processus de construction





Redémarrez la cible : Il est nécessaire ici de modifier le script de démarrage de la cible pour pouvoir accéder au service ssh depuis votre hôte en utilisant une redirection de port. Copiez le script start-qemu.sh et renommez le start-qemu-net.sh

~/Documents/buildroot/output/images$ cp start-qemu.sh start-qemu-net.sh

* Modifiez le script start-qemu-net.sh comme ci-dessous :

#!/bin/sh IMAGE\_DIR="${0%/\*}/"

if [ "${1}" = "serial-only" ]; then EXTRA\_ARGS='-nographic'

else

EXTRA\_ARGS='-serial stdio'

fi

export PATH="/home/marco-virt/Documents/buildroot/output/host/bin:${PATH}" exec qemu-system-arm \

-M versatilepb -kernel ${IMAGE\_DIR}/zImage \

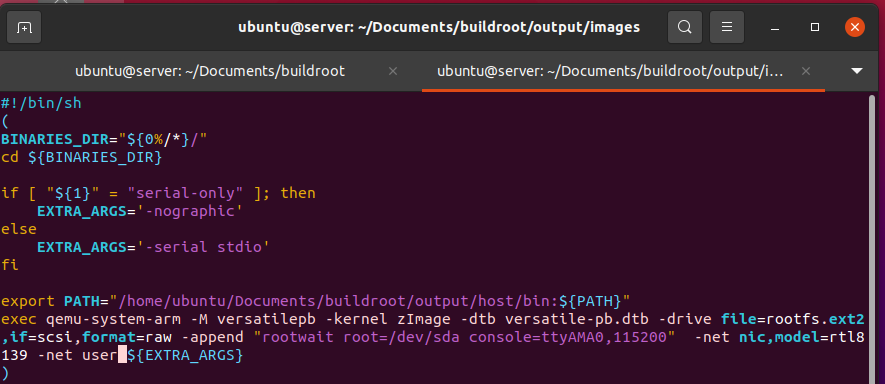
-dtb ${IMAGE\_DIR}/versatile-pb.dtb \

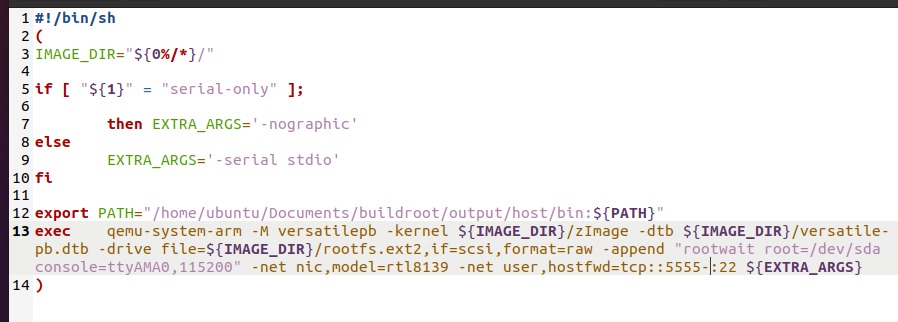
-drive file=${IMAGE\_DIR}/rootfs.ext2,if=scsi,format=raw \

-append "rootwait root=/dev/sda console=ttyAMA0,115200" \

-net nic,model=rtl8139 -net user,hostfwd=tcp::5555-:22 \

${EXTRA\_ARGS}





* + - * Observez les effets de la reconstruction sur les modifications réalisées antérieurement (le prompt ?).

La taille du noyau est devenue plus grande que la première fois.

Temps de démarrage est plus long a mis 50 seconds, le deuxième environs 30 seconds.

Assurez vous de la nécessité de fournir le bon mot de passe root pour accéder au système.

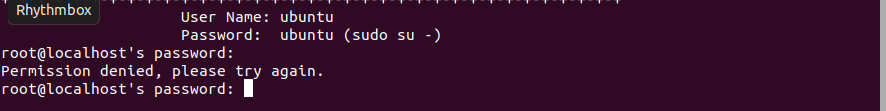




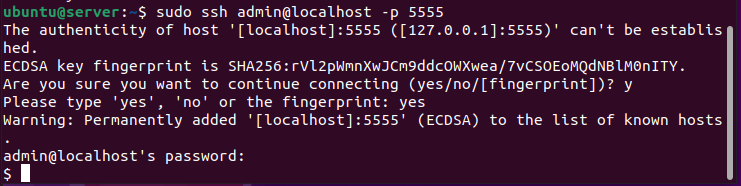
Accédez à la cible depuis votre PC hôte via une session ssh. Assurez-vous que root ne puisse pas y accéder directement.

ssh-admin@localhost -p 5555

**Compte root**



**Compte admin**



Acceptez la clé de chiffrement. Il sera nécessaire de la supprimer lors des futures tentatives de connexion après chaque reconstruction du système.

**Pourquoi root ne devrait jamais pouvoir ouvrir une connexion distante directement.**

Root ne peut pas accéder à distance parce que root est l’utilisateur par défaut dans le système et son accès se fait directement dans le logging normal,

On a besoin de personaliser le fichier **sshd\_config** sur la carte avec "PermitRootLogin yes" défini sur capable de ssh en tant qu'utilisateur root.

et les utilisateurs autorisés à accéder sont définis dans le fichier **adduser.txt** et si on autorise un accès à distance.

### Persistance des modifications sur rootfs

Les modifications réalisées directement sur cible à partir de l’hôte fonctionnent très bien pour un système unique et final. Cependant, lors la prochaine reconstruction du système, ces changements auront disparu car ils ne sont pas présents dans la configuration de buildroot.

Il est important que le processus de construction reste entièrement reproductible, si l’on veut s’assurer que la prochaine version inclura les configurations personnalisées.

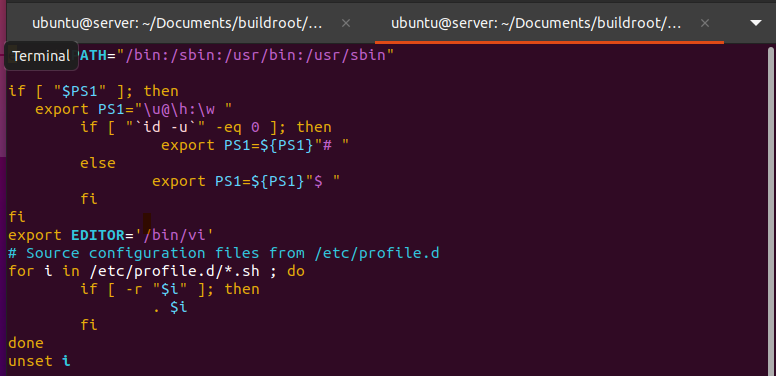
Pour ce faire, le plus simple est d'utiliser le mécanisme de recouvrement du rootfs de Buildroot (overlay mechanism).

Cette substitution de fichiers est spécifique au projet en cours. Il faut donc créer un répertoire personnalisé pour ce projet dans la source de buildroot pour la cible visée dans le dossier :

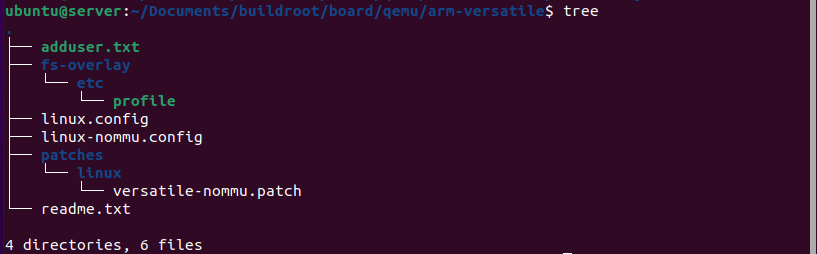
**board/<manufacturer>/<boardname>/fs-overlay/**Créez le dossier de recouvrement fs-overlay et copiez-y le script précédents assurant la bonne configuration du prompt.

~/Documents/buildroot $ cd board/qemu/arm-versatile/

~/Documents/buildroot/board/qemu/arm-versatile $ mkdir fs-overlay

Fichier profile  


Contenu du dossier fs-factory



* Indiquez la présence de ce dossier à **buildroot.**

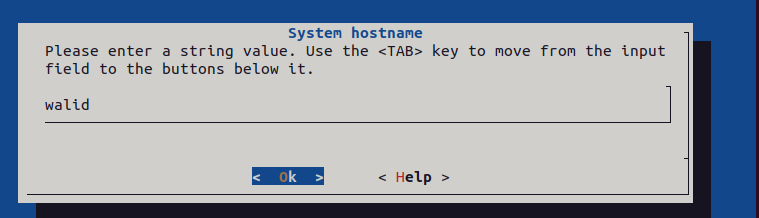
### make menuconfig → System configuration → Root filesystem overlay directories

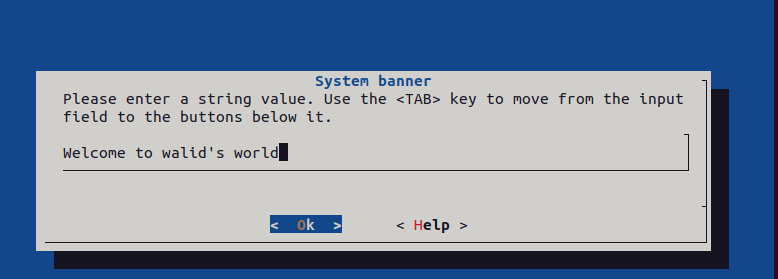
### 

* Ajoutez au système un nom d'hôte et une bannière d'accueil.

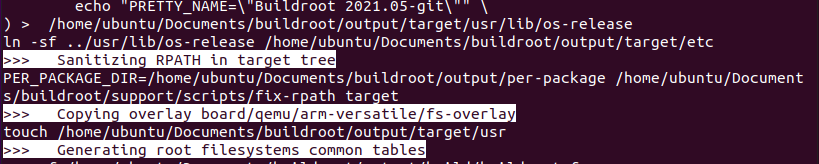
### make menuconfig → System configuration → System hostname

**→ System banner**





* Relancez le processus de construction.
* Redémarrez la cible et observez les modifications réalisées.



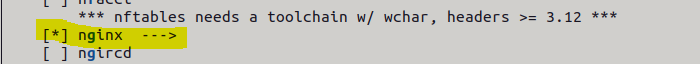


### Serveur web

Plusieurs serveur web sont disponible dans buildroot : Apache, lighttpd, nginx. Nous choisirons ici d’installer nginx :

### make menuconfig → Target packages → Networking applications

**→ nginx**

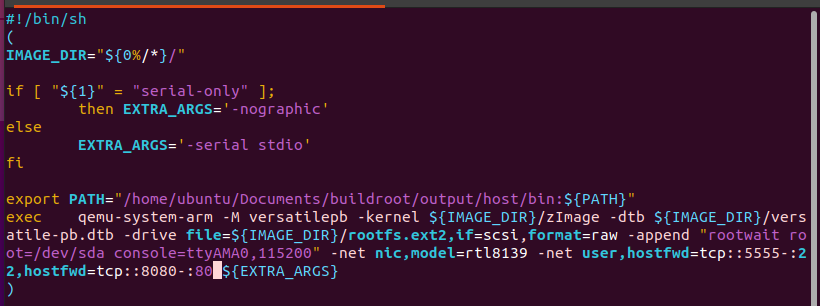


* + - * Relancez le processus de construction.

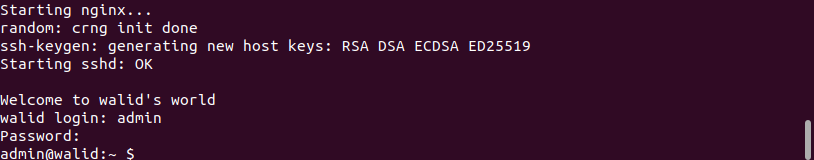


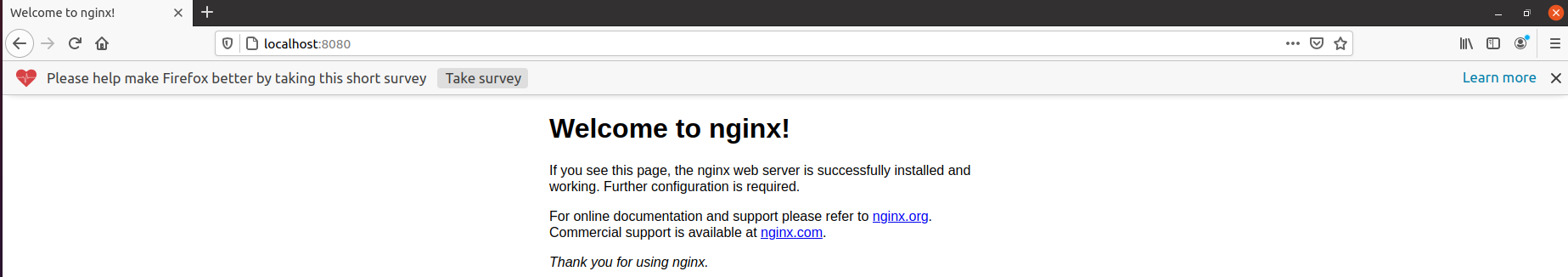
Modifiez le script de démarrage start-qemu-net.sh pour ajouter une redirection du port 80 vers le port 8080

-net nic,model=rtl8139 -net user,hostfwd=tcp::5555-:22,hostfwd=tcp::8080-:80 \

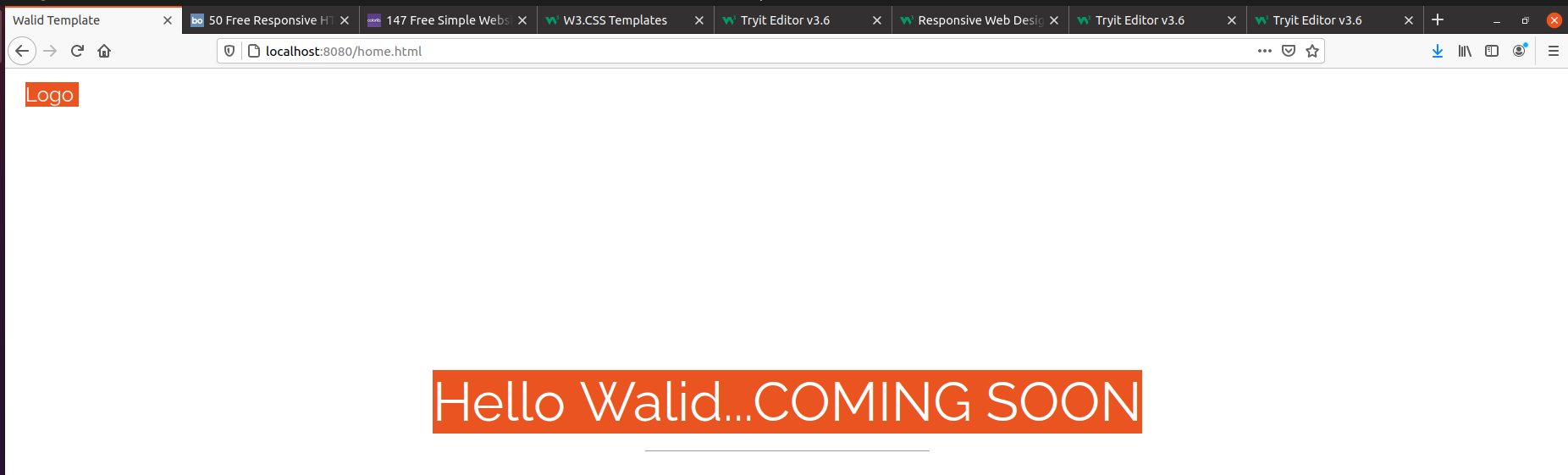


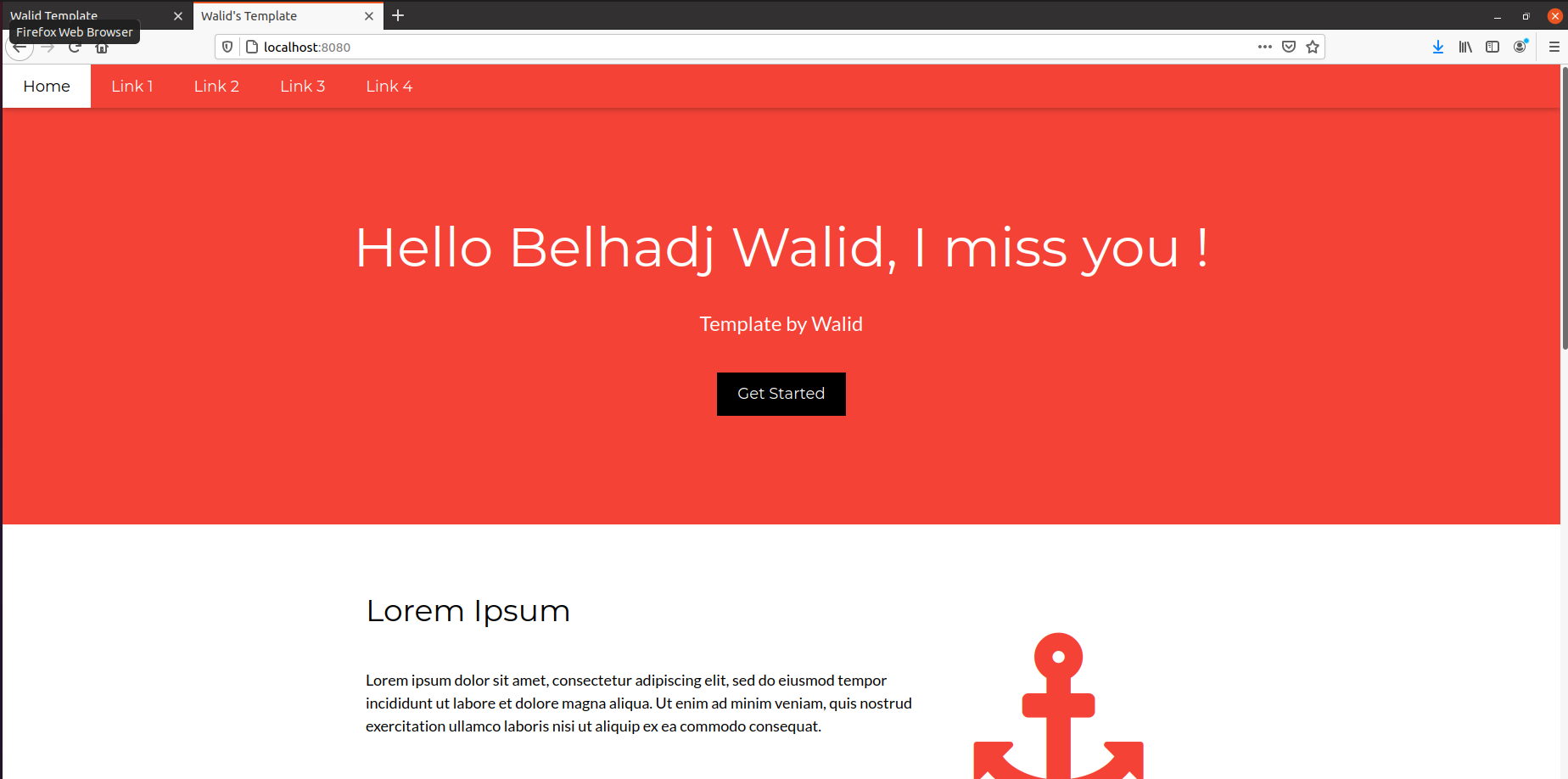
Redémarrez la cible accédez au service web depuis votre hôte en utilisant l’URL [**http://localhost:8080**](http://localhost:8080)



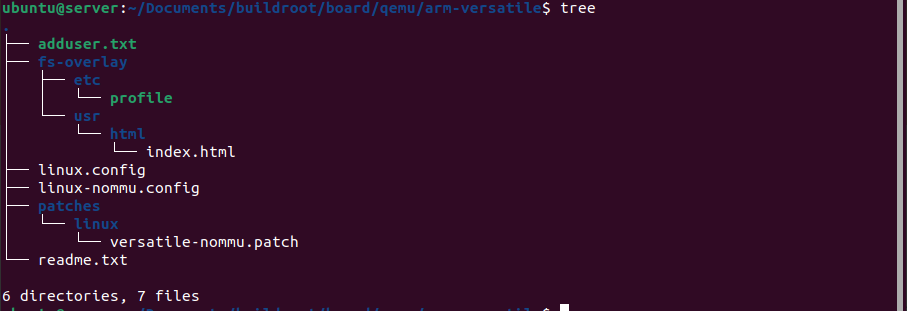


* + - * Le dossier racine du serveur web se trouve dans /usr/html. Proposez une page index.html qui contient vos nom et prénom.

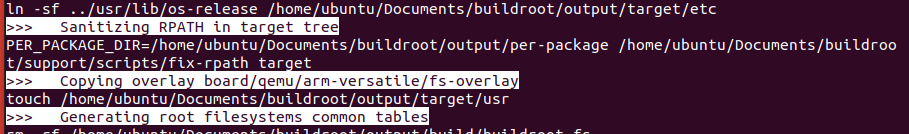


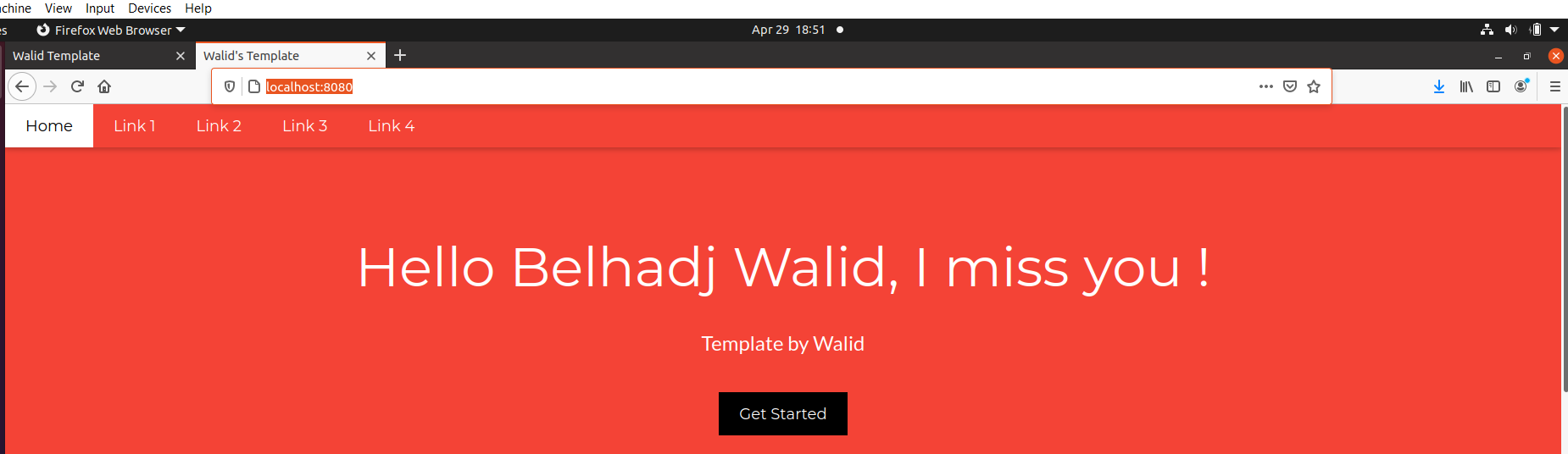


Modifiez l’arborescence du dossier fs-overlay pour que la prochaine reconstruction contienne votre page web



**make**



Sur le navigateur : la page a été bien mise à jour   


# Image finale