

Rappels sur Yocto Project

Christophe BLAESS

christophe.blaess@logilin.fr

<https://www.blaess.fr/christophe/>



Ingénierie et formations sur Linux et les logiciels libres
<https://www.logilin.fr>

| | |
|--|-----------|
| Linux embarqué et <i>build systems</i>..... | 3 |
| Comparaison entre Yocto Project et Buildroot..... | 4 |
| Arborescence de travail recommandée..... | 5 |
| Emploi du fichier `site.conf` | 7 |
| Travaux pratiques : production d'une image standard..... | 8 |
| Rappels des commandes principales..... | 9 |
| Contenu typique d'un layer pour un projet embarqué..... | 11 |
| Travaux pratiques : 1 – Utilisation d'un layer personnalisé..... | 12 |
| Travaux pratiques : 2 – Utilisation d'un layer téléchargé..... | 13 |

Ce support de formation est distribué sous licence **Creative Commons 4.0**



(Attribution - Partage dans les mêmes conditions).

Vous êtes libres de copier et partager ce document, en mentionnant son origine. Si vous l'intégrez dans un contenu plus vaste, ce dernier devra être distribué avec les mêmes droits.

Ce cours a été rédigé en utilisant des logiciels libres sur système d'exploitation Linux :

- *LibreOffice Writer* pour le support et la mise en page
- *Gimp* pour les images bitmap
- *Excalidraw* (en ligne) pour les schémas vectoriels.

Yocto Project avancé

YPA v. 1.2

<https://www.blaess.fr/christophe/>

<https://www.logilin.fr>

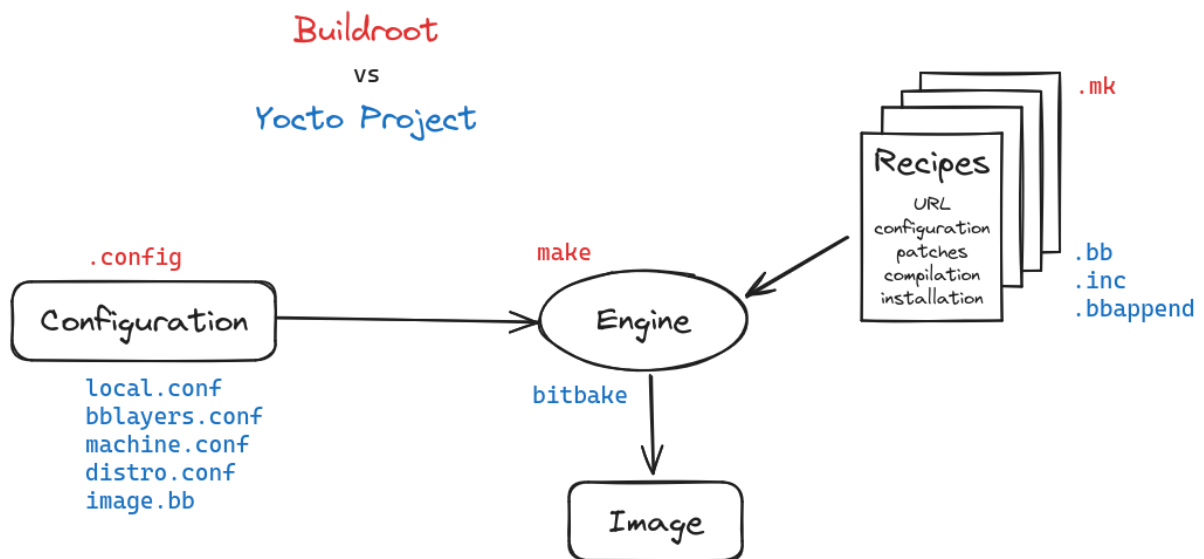
Linux embarqué et *build systems*

Yocto Project est un **build system**, un environnement logiciel qui produit une image complète d'un système Linux prêt à être installé sur une carte à microprocesseur.

Yocto Project repose sur un moteur de compilation nommé **bitbake** qui en fonction du contenu de fichiers de configuration sélectionne des **recettes** (*recipes*), c'est-à-dire des fichiers décrivant comment produire ou assembler des packages logiciels.

Les recettes sont des fichiers avec l'extension `.bb`, qui peuvent inclure des fichiers d'extension `.inc` et être amendés par des fichiers d'extension `.bbappend`, lors du *parsing* réalisé par `bitbake`.

Comparaison entre Yocto Project et Buildroot



Arborescence de travail recommandée

Les recettes sont regroupées par fonctionnalités connexes dans des layers.

Un **layer** est un répertoire dont le nom commence par ``meta-``.

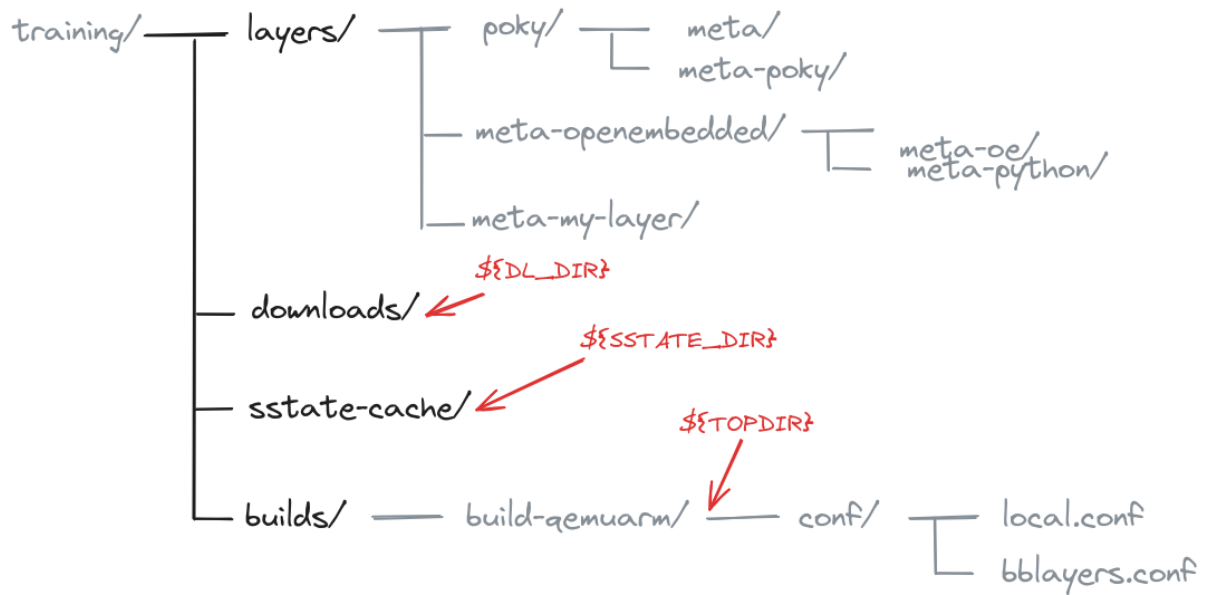
Il contient des recettes dans des sous-répertoires ``recipes-...`` ainsi que des fichiers de configuration.

Dans le fichier ``layer.conf`` est indiqué la **priorité** du layer.

On commence souvent un projet basé sur Yocto en partant de **Poky**, une sorte de “*starter pack*” regroupant ``bitbake`` (et d’autres outils), des recettes de départ et des fichiers de configuration

Poky se télécharge avec ``git clone`` depuis <https://git.yoctoproject.org/poky>

Recommended Yocto Project Work Tree



Emploi du fichier `site.conf`

Les principaux fichiers de configuration pris en compte par `bitbake` sont, dans l'ordre :

- `conf/bblayers.conf` qui définit la variable `BBLAYERS`,
- `conf/layer.conf` pour chaque layer indiqué dans `${BBLAYERS}`,
- `conf/site.conf` (qui n'existe pas par défaut),
- `conf/local.conf` dans lequel on définit entre autres `MACHINE`, `SDKMACHINE` et `DISTRO`.
- `conf/machine/${MACHINE}.conf`
- `conf/machine-sdk/${SDKMACHINE}.conf`
- `conf/distro/${DISTRO}.conf`

Le fichier `conf/local.conf` est normalement partagé entre les développeurs réalisant le même *build*, alors que `conf/site.conf` est spécifique pour un *build* à un emplacement donné.

Travaux pratiques : production d'une image standard

- ➔ Préparez les répertoires principaux ``layers`` et ``builds`` conformément à la disposition recommandée (les autres seront créés automatiquement).
- ➔ Téléchargez (dans le sous-répertoire global ``layers``) la branche ``scarthgap`` de Poky et extrayez le tag Git ``yocto-5.0.7``.
- ➔ En vous plaçant au sommet de l'arborescence, initialisez un répertoire de *build* avec la commande :

```
$ source layers/poky/oe-init-build-env builds/build-qemuarm
```

- ➔ Éditez le fichier ``conf/site.conf`` pour remplir les variables :

```
DL_DIR = "${TOPDIR}/../../../../downloads"
SSTATE_DIR = "${TOPDIR}/../../../../sstate-cache"
```

- ➔ Éditez le fichier ``conf/local.conf`` pour remplir ``MACHINE = "qemuarm"``

- ➔ Lancez un premier *build* de l'image ``core-image-base`` puis vérifiez qu'il fonctionne en exécutant

```
$ runqemu nographic
```

Solution :

```
$ mkdir layers
$ cd layers/
```

```
$ git clone https://git.yoctoproject.org/poky -b scarthgap
$ cd poky
$ git checkout yocto-5.0.7
$ cd ../../
```

```
$ mkdir builds
```

```
$ source layers/poky/oe-init-build-env builds/build-qemuarm
```

```
$ nano conf/site.conf
|| DL_DIR = "${TOPDIR}/../../../../downloads"
|| SSTATE_DIR = "${TOPDIR}/../../../../sstate-cache"
```

```
$ nano conf/local.conf
|| MACHINE = "qemuarm"
```

```
$ bitbake core-image-base
$ runqemu nographic
[...]
```

```
Poky (Yocto Project Reference Distro) 5.0.7 qemuarm /dev/ttyAMA0
qemuarm login: root
root@qemuarm:~#
```


Rappels des commandes principales

\$ source layers/poky/oe-init-build-env builds/my-build

Crée le répertoire de *build* indiqué s'il n'existe pas, et initialise les variables d'environnement nécessaires pour accéder aux outils fournis par Poky.

\$ bitbake <package-name>

Lance le *build* pour obtenir le package indiqué (souvent une recette d'image).

\$ bitbake -e <package-name>

Affiche sur la sortie standard toutes les variables d'environnement renseignées après parcours des recettes nécessaires à la production du package.

\$ bitbake -k <package-name>

Lance le *build* en continuant le plus longtemps possible en cas d'erreur.

\$ bitbake -g -u taskexp <package-name>

Explore le graphe de dépendance du package indiqué.

\$ bitbake --runall=fetch <package-name>

Télécharge toutes les sources nécessaires pour produire le package.

\$ bitbake -s

Affiche la liste de toutes les recettes des *layers* connus avec leurs versions préférées.

\$ bitbake-layers show-layers

Affiche la liste des layers pris en compte par `bitbake`. Cette liste est contenue dans le fichier `conf/bblayers.conf` à partir du répertoire de *build*.

\$ bitbake-layers create-layer <layer-path>

Crée un nouveau layer pour y stocker des recettes. Des paramètres par défaut sont enregistrés dans le fichier `*<layer-path>/conf/layer.conf*`.

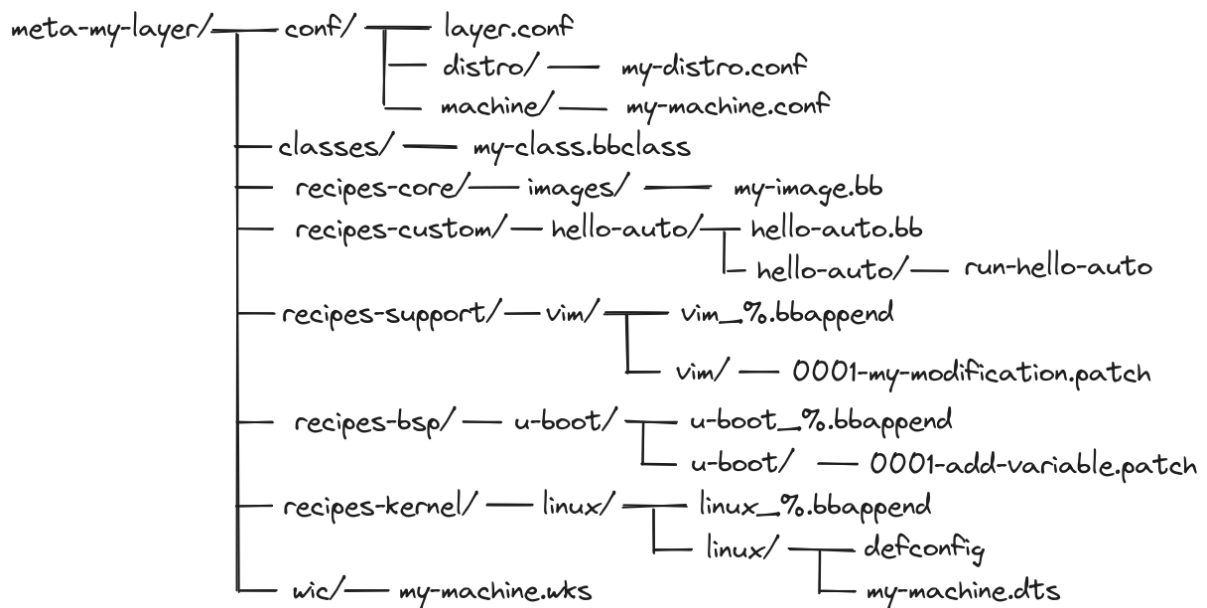
\$ bitbake-layers add-layer <layer-path>

Ajoute un layer existant à la liste de ceux pris en considération par `bitbake`.

\$ runqemu [nographic]

Lance Qemu pour exécuter l'image compilée (avec ou sans environnement graphique).

Contenu typique d'un layer pour un projet embarqué

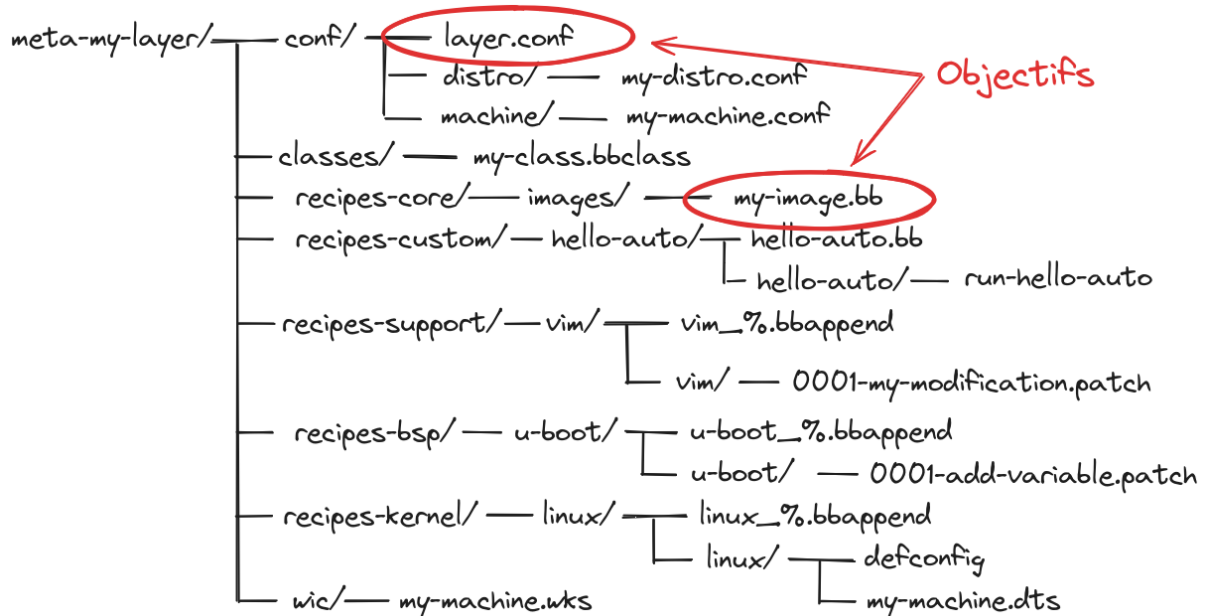


En pratique on sépare souvent les éléments de configuration liés au matériel (*kernel*, *bootloader*, etc.) de ceux concernant l'espace applicatif et le code métier.

Travaux pratiques : 1 – Utilisation d'un layer personnalisé

- Créez un layer `meta-my-layer` dans le sous-répertoire global `layers` et ajoutez ce layer à la liste de ceux pris en compte pour le *build*.
- Créez dans ce layer une recette d'image `my-image.bb` qui hérite de la classe `core-image` et ajoute dans `IMAGE_INSTALL` le package `os-release`.

(On peut s'inspirer de `layers/poky/meta/recipes-core/images/core-image-base`)



- Lancez le build de l'image `my-image` puis redémarrez Qemu et vérifiez la présence de `/etc/os-release` sur la cible.

Solution

```
$ bitbake-layers create-layer ../../layers/meta-my-layer
$ bitbake-layers add-layer ../../layers/meta-my-layer
$ mkdir -p ../../layers/meta-my-layer/recipes-core/images
$ nano ../../layers/meta-my-layer/recipes-core/images/my-image.bb
| inherit core-image
| IMAGE_INSTALL:append = " os-release"
$ bitbake my-image
$ runqemu nographic
[...]
Poky (Yocto Project Reference Distro) 5.0.7 qemuarm /dev/ttyAMA0
qemuarm login: root
root@qemuarm:~# cat /etc/os-release
ID=poky
NAME="Poky (Yocto Project Reference Distro)"
VERSION="5.0.7 (scarthgap)"
[...]
```

Travaux pratiques : 2 – Utilisation d'un layer téléchargé

- Téléchargez (dans le sous-répertoire global ``layers``) la branche `scarthgap` de ``meta-openembedded`` depuis <https://git.openembedded.org/meta-openembedded>
- Ajoutez le layer ``meta-oe`` de ``meta-openembedded`` dans la liste des répertoires pris en compte par ``bitbake``.
- Ajoutez les packages ``nano``, ``usleep`` et ``vim`` dans la variable ``IMAGE_INSTALL``.
- Relancez le *build*, puis vérifiez la présence des packages sur la cible.

Solution

```
$ cd ../../layers
$ git clone https://git.openembedded.org/meta-openembedded -b scarthgap
$ cd ../builds/build-qemuarm/
$ bitbake-layers add-layer ../../layers/meta-openembedd ed/meta-oe/
$ nano ../../layers/meta-my-layer/recipes-core/images/my-image.bb
| [...]
| IMAGE_INSTALL:append = " nano"
| IMAGE_INSTALL:append = " usleep"
| IMAGE_INSTALL:append = " vim"
|
$ bitbake my-image
```

