Yocto - devtool - Ansible
La dernière recette de ma grand-mère

Mickaël Tansorier

Présenstation sur le fonctionnement de Yocto et d'outils pratiques.

Objectif de la présentation

- Présenter Yocto
- Démonstration concrète sur Rapsberry Pi
- Présentation d'outils utiles
  - `devtool`
  - `Ansible`

Plan

D'où vient ce nom ?

Definition

Yocto est un préfixe représentant 10^{-24} unités (SI)

Qu'est ce qu'est vraiment Yocto ?

Yocto est un outil qui répond au besoin de générer une distribution **Linux embarqué** pour un matériel **dédié**.

Pourquoi Yocto existe ?

Ce projet s'est basé sur l'outil **OpenEmbedded** pour voir le jour.

En effet il y avait une volonté de pouvoir moduler les applications sur **différents matériels** sans avoir à investir dans un nouveau développement.

Des développeurs et la Fondation Linux se sont unis pour proposer une mécanique qui fasse abstraction du matériel, et ainsi rendre réutilisables les développements déjà effectués.

Depuis 2010 ce projet continue sa route !

Présentation de Yocto

Présentation de Yocto

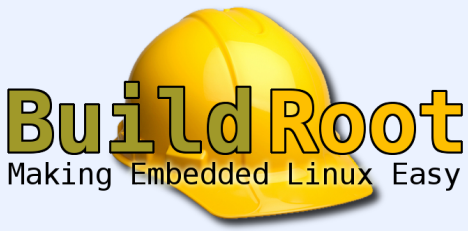
Pas si vite …

Avant tout, un peut de contexte.

- Est-ce que Yocto est le seul outils qui existe ?
- Qu'est-ce qu'il a de plus que les autres ?
- Comment c'est architecturé ?

D'autres outils permettent de créer des distributions Linux pour les systèmes embarqués.

Buildroot



Buildroot est un outil qui ressemble à un jeu de makefile, capable de générer tous les éléments pour démarrer un système sous Linux.

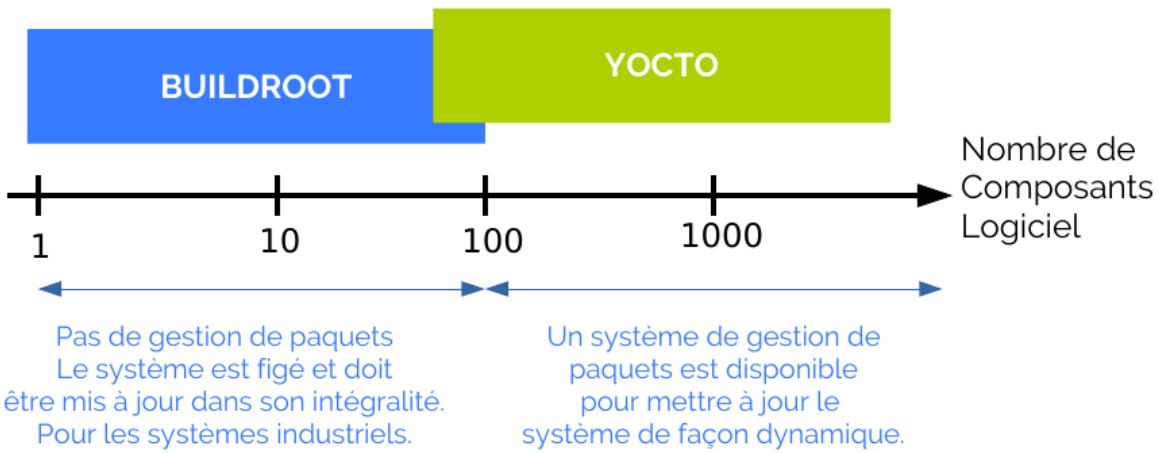
Linux From Scratch

Linux from scratch est un projet qui décrit pas à pas les instructions pour construire un système Linux. Très instructif mais fastidieux !

OpenWrt

OpenWrt est un système libre et issu de Buildroot. Très orienté réseau, il offre la possibilité de gérer les paquets ipk.

Yocto et Buildroot sont deux outils très proches mais avec fonctionnalité qui diffères en fonction des besoins



La différenciation de l'**architecture matérielle** de l'**application logicielle** de la cible

- `MACHINE` : définit l'architecture matérielle
- `DISTRO` : définit la distribution à générer

Une **communauté** active

- Nouvelle version tous les 6 mois
- 1 version de dev, 3 stables, le reste en communauté
- Channel IRC actif

De la **documentation** bien fournie

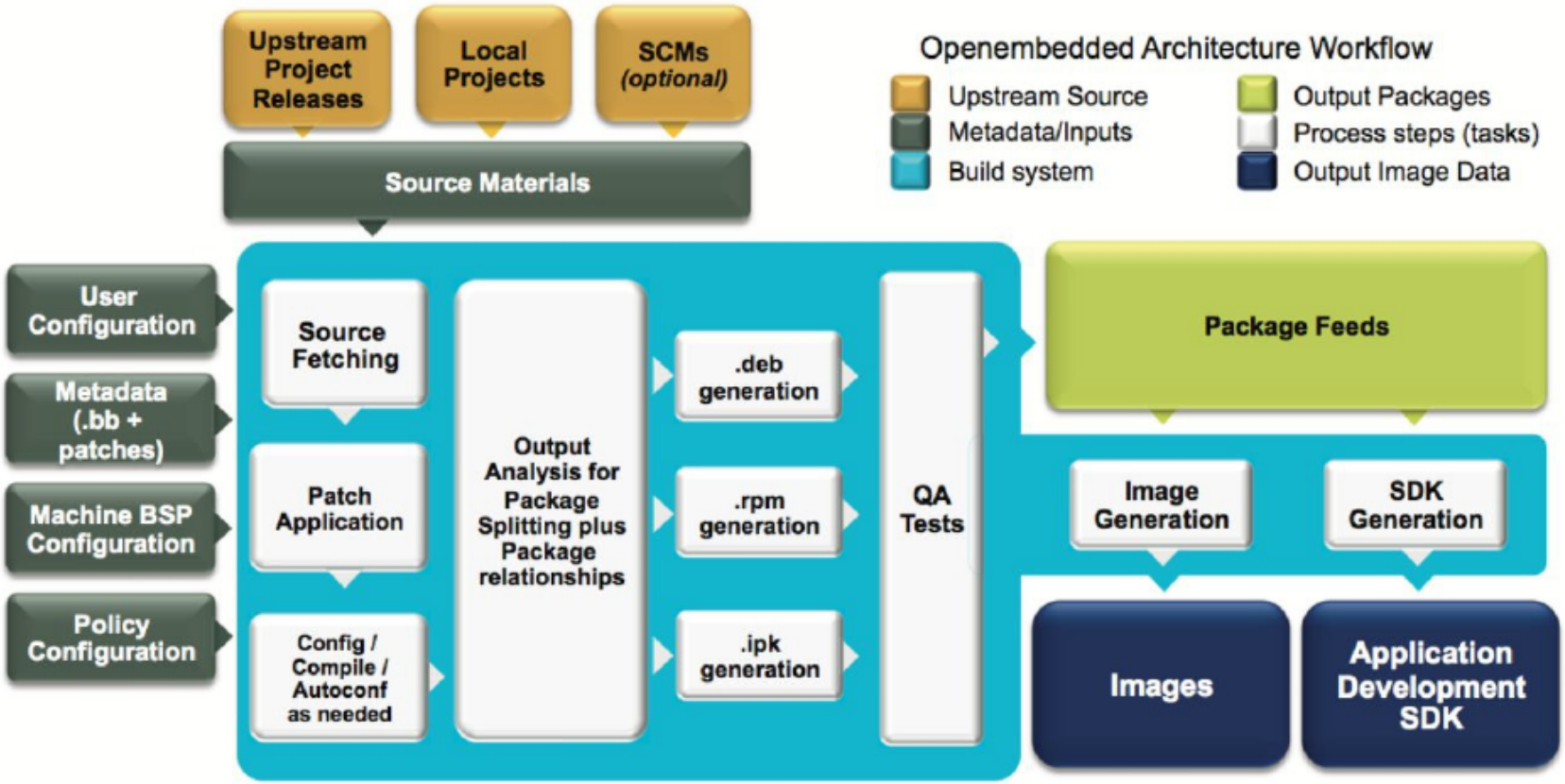
- Doc classique
- Vidéo

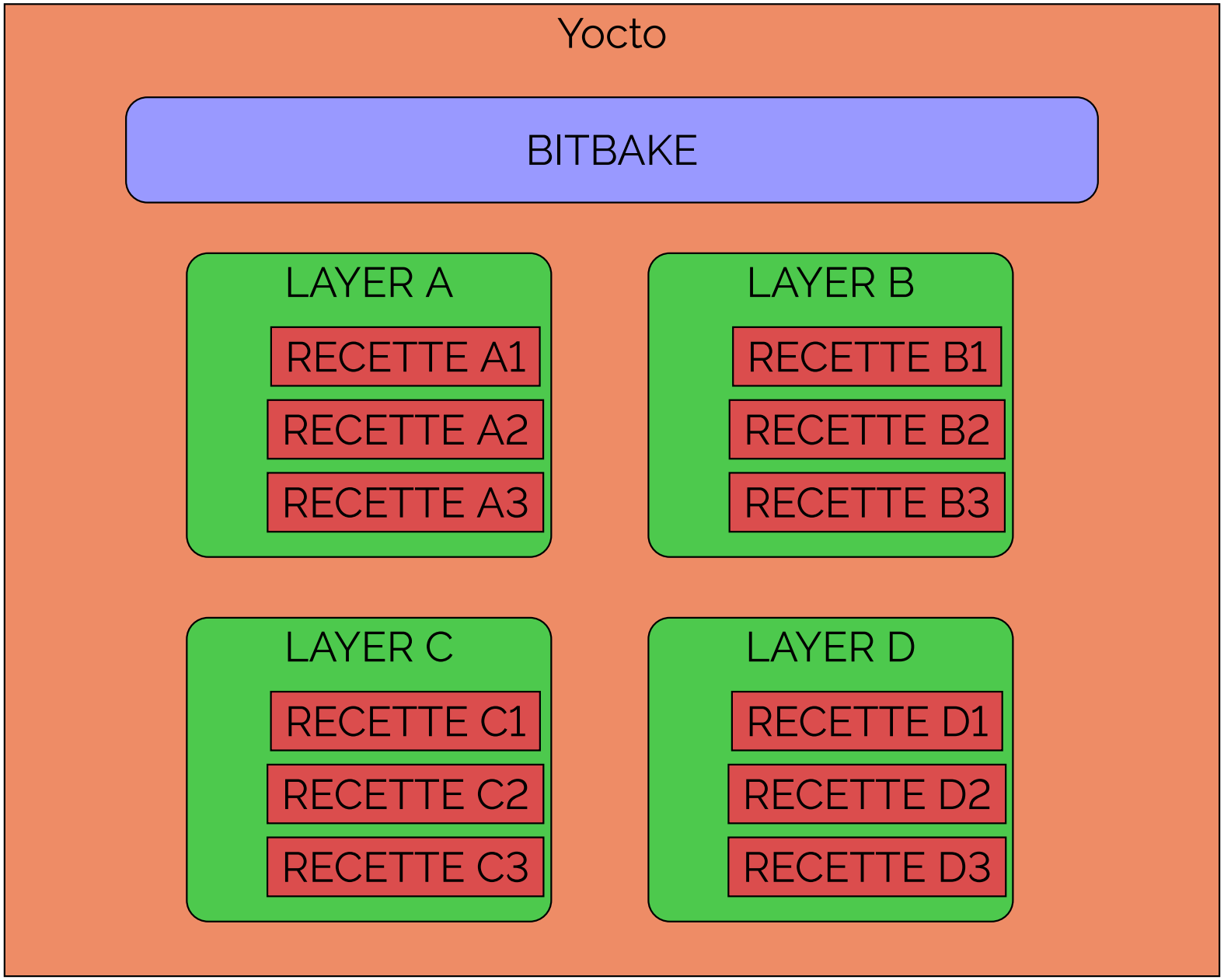
Des **outils** puissants

- `devtool`
- `ipk/opkg`

- Yocto est gourmand en ressources, une configuration minimale de 50Go de disque dur, un CPU à 1,6GHz et 8Go de RAM est recommandée.
- Plusieurs distributions Linux supportent Yocto : Ubuntu, Fedora, Debian, OpenSuse, CentOS.
- Le projet Yocto produit une nouvelle version majeure tous les 6 mois environ.
- Elle porte généralement un nom associé à un numéro de version. ex : Morty (2.2), Pyro (2.3), Rocko (2.4), Sumo (2.5), ...

Workflow


Openembedded Architecture Workflow
Upstream Source
Metadata/Inputs
Build system
Output Packages
Process steps (tasks)
Output Image Data
Upstream Project Releases
Local Projects
SCMs (optional)
Source Materials
User Configuration
Metadata (.bb + patches)
Machine BSP Configuration
Policy Configuration
Source Fetching
Patch Application
Config / Compile / Autoconf as needed
Output Analysis for Package Splitting plus Package relationships
.deb generation
.rpm generation
.ipk generation
QA Tests
Package Feeds
Image Generation
SDK Generation
Images
Application Development SDK


Yocto
BITBAKE
LAYER A
RECETTE A1
RECETTE A2
RECETTE A3
LAYER B
RECETTE B1
RECETTE B2
RECETTE B3
LAYER C
RECETTE C1
RECETTE C2
RECETTE C3
LAYER D
RECETTE D1
RECETTE D2
RECETTE D3

Quelques layers générique communautaire

- meta
- meta-openembedded
  - meta-oe
  - meta-networking
  - meta-python
  - meta-gnome
- meta-poky

D'autre plus spécifique

- meta-raspberrypi
- meta-intel
- meta-xfce
- meta-qt5

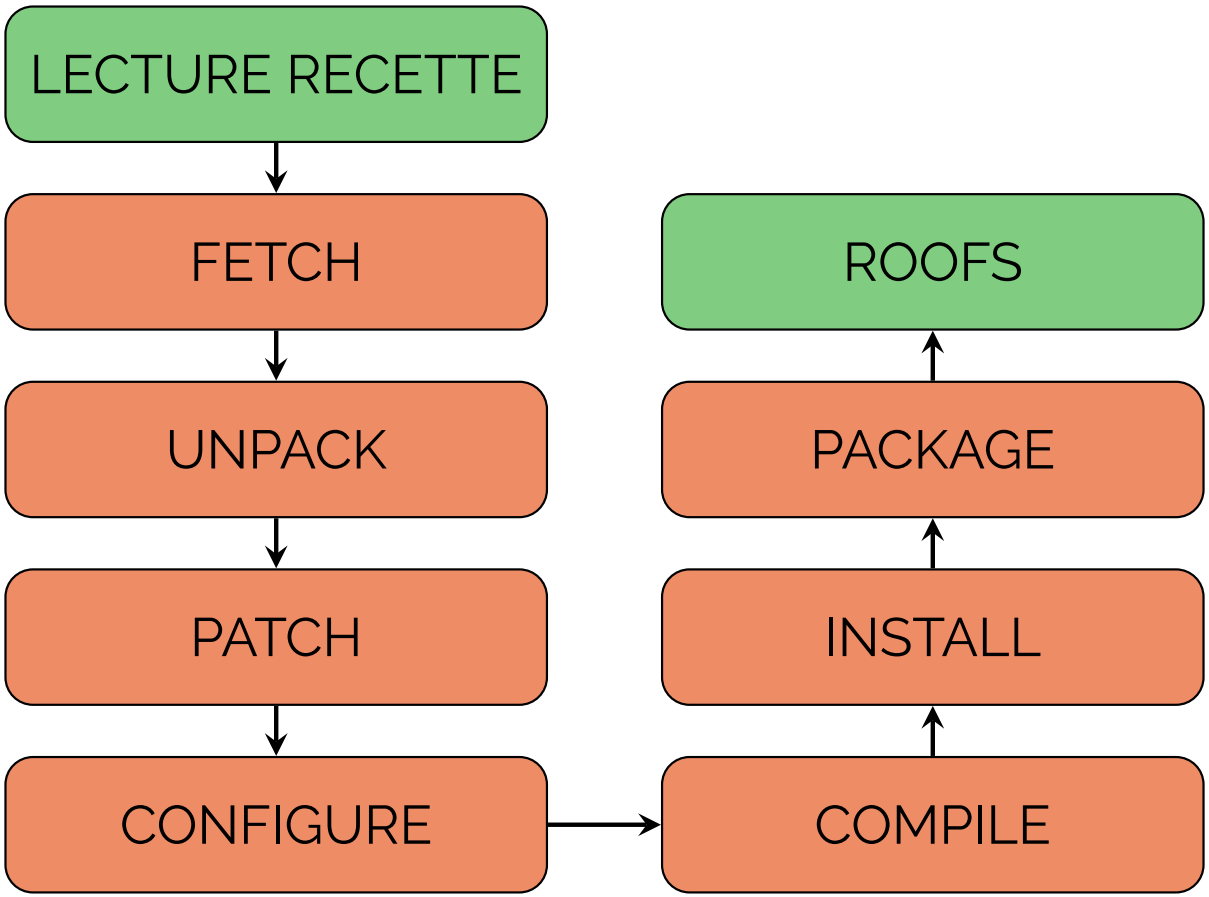
Avant de passer aux recettes, qui fait le travail dans Yocto ?

bitbake

bitbake c'est quoi ?

- Un moteur d'exécution de tâches écrite en Python
- Fonctionne en ligne de commande
- Exécute automatiquement les tâches nécessaires à la fabrication de la cible fournie

bitbake


LECTURE RECETTE
FETCH
UNPACK
PATCH
CONFIGURE
COMPILE
INSTALL
PACKAGE
ROOFS

recette

À quoi ça ressemble une recette ?

```
1 DESCRIPTION = "Exemple d'une description d'une recette"
2 LICENSE = "MIT"
3 LIC_FILES_CHKSUM = "file://${COMMON_LICENSE_DIR}/MIT;md5=0835ade698e0bcf8506ecda2f7b4f302"
4
5 SRC_URI = "git://github.com/exemple/exemple.git \
6            file://un-patch.patch \
7            file://un-fichier \
8 "
9
10 SRCREV = "2693ca21cee8a729d74682fd86a4818f2b050228"
11
12 S = "${WORKDIR}/git"
13
14 do_configure() {
15     # Commentaire d'une fonction pour spécifier la config
16 }
17
18 EXTRA_OECONF = "--une-option"
19
20 do_install() {
21     install -d ${D}${bindir}/un-dossier
22     install -m 0755 ${S}/un-binaire-construit ${D}${bindir}/un-dossier/un-binaire-construit
23 }
24
25 FILES_${PN} = "${bindir}/un-dossier/un-binaire-construit"
```

Comment crée son image spécifique pour une carte donnée ?

Yocto à la particularité de bien séparer la **distribution** de l'**architecture matériel**.

Les architectures matériel

- ARM
- x86
- x86-64
- PowerPC
- MIPS

Les cartes associés

- Raspberry Pi (différent versions)
- Beaglebone
- intel-core2-32

Les différents BSP sont répertoriés sur le site de yoctoproject : `https://www.yoctoproject.org/downloads/bsps`

Le paramétrage de la `DISTRO` et de la `MACHINE` se fait en local.

`$POKY/build/conf/local.conf`

```
# This sets the default machine to be qemux86 if no other machine
    is selected:
MACHINE ??= "qemux86"
# Default distro:
DISTRO ?= "poky"

# Mes parametres
MACHINE = "raspberrypi3-64"
DISTRO = "distromeetup"
```

TP Raspberry Pi

TP Raspberry Pi

Les étapes :

1. Installer l'environnement de développement
2. Construire une distribution générique
3. Créer sa propre distribution
4. Tester son image

On se base sur la dernière version de Yocto stable disponible, c'est à dire `rocko` qui est sortie en octobre 2017. (la prochaine est en avril 2018)

```
$ git clone git://git.yoctoproject.org/poky -b rocko
$ cd poky
$ git clone git://git.yoctoproject.org/meta-raspberrypi -b rocko
$ git clone git://git.openembedded.org/meta-openembedded -b rocko
```

Pour construire son image il faut d'abord :

- Sourcer l'environnement pour bitbake

```
$ . oe-init-build-env
```

  Cela nous créer un dossier `build` dans lequel tout va se passer.

- Paramétré la `MACHINE` et la `DISTRO`
  Ces variables sont écrites dans le fichier `local.conf`.

Il reste plus qu'à modifier `$POKY/build/conf/local.conf`

```
# Mes parametres
MACHINE = "raspberrypi3-64"
```

et ajouter le layer manquant dans `$POKY/build/conf/bblayers.conf`

```
BBLAYERS += " \
    ${TOPDIR}/../meta-raspberrypi \
"
```

Les chemains sont en général absolue comme : `/home/username/path/to/project/poky/meta-raspberrypi`

Et enfin lancer la construction de l'image avec

```
$ bitbake core-image-minimal
```

Yocto est là pour vous aider à construire votre image.
En effet si l'on ne rajoute seulement `meta-raspberrypi` dans `bblayers.conf` on obtiens l'erreur suivante :

meta-python

```
ERROR: ParseError at /home/ubuntu/meetup/poky/meta-raspberrypi/
    recipes-devtools/python/rpio_0.10.0.bb:9: Could not inherit
    file classes/pypi.bbclass
```

Il faut donc rajouter `meta-openembedded/meta-python`

meta-oe

```
ERROR: Layer 'meta-python' depends on layer 'openembedded-layer',
     but this layer is not enabled in your configuration
```

De plus dans `meta-python/conf/layer.conf` on a

```
LAYERDEPENDS_meta-python = "core openembedded-layer"
```

Il faut donc rajouter les layers qui vont bien dans `bblayers.conf`

```
BBLAYERS += " \
    ${TOPDIR}/../meta-raspberrypi \
    ${TOPDIR}/../meta-openembedded/meta-python \
    ${TOPDIR}/../meta-openembedded/meta-oe \
"
```

On peut maintenant construire et tester une image

Étape suivante :

Créer sa propre distribution

Pour créer sa propre distribution il est préférable de créer son propre layer

```
$ cd $POKY
$ mkdir -p meta-meetup/conf
```

Il faut déclarer la layer avec `./conf/layer.conf`

```
# We have a conf and classes directory, add to BBPATH
BBPATH .= ":${LAYERDIR}"

# We have recipes-* directories, add to BBFILES
BBFILES += "${LAYERDIR}/recipes-*/*/*.bb \
            ${LAYERDIR}/recipes-*/*/*.bbappend"

BBFILE_COLLECTIONS += "meetup"
BBFILE_PATTERN_meetup = "^${LAYERDIR}/"
BBFILE_PRIORITY_meetup = "10"
```

Il faut maintenant l'ajouter dans
`$POKY/build/conf/bblayers.conf`

```
BBLAYERS += " \
    ${TOPDIR}/../meta-raspberrypi \
    ${TOPDIR}/../meta-openembedded/meta-python \
    ${TOPDIR}/../meta-openembedded/meta-oe \
    ${TOPDIR}/../meta-meetup \
"
```

Créer sa distro avec `./conf/distro/distromeetup.conf`

```
# Distribution base sur poky
require conf/distro/poky.conf

DISTRO = "distromeetup"
DISTRO_NAME = "Distro meetup example"
DISTRO_VERSION = "0.1"

# Ajout d'option pour la distribution
DISTRO_FEATURES_append = " systemd"
VIRTUAL-RUNTIME_init_manager = "systemd"

# Utilisation seulement du paquetage ipk
PACKAGE_CLASSES = "package_ipk"
```

Si on souhaite utiliser la nouvelle distribution il faut ajouter dans `$POKY/build/conf/local.conf`

```
MACHINE = "raspberrypi3-64"
DISTRO = "distromeetup"
```

Créer son image et choisir ce que l'on met dedans ?

Par defaut la distro héritant de poky contiens

```
DISTRO_FEATURES = "${DISTRO_FEATURES_DEFAULT} ${
    DISTRO_FEATURES_LIBC} ${POKY_DEFAULT_DISTRO_FEATURES}"
```

Avec dans chaques variables

```
DISTRO_FEATURES_DEFAULT="acl alsa argp bluetooth ext2 irda
    largefile pcmcia usbgadget us bhost wifi xattr nfs zeroconf
    pci 3g nfc x11"

DISTRO_FEATURES_LIBC="ipv4 ipv6 libc-backtrace libc-big-macros
    libc-bsd libc-cxx-tests l ibc-catgets libc-charsets
    libc-crypt libc-crypt-ufc libc-db-aliases libc-envz
    libc-fcvt libc-fmtmsg libc-fstab libc-ftraverse
    libc-getlogin libc-idn libc-inet-anl libc-libm libc-locales
    libc-locale-code libc-memusage libc-nis libc-nsswitch
    libc-rcmd libc-rtld-debug libc-spawn libc-streams
    libc-sunrpc libc-utmp libc-utmpx libc-wordexp
    libc-posix-clang-wchar libc-posix-regexp
    libc-posix-regexp-glibc libc-posix-wchar-io"

POKY_DEFAULT_DISTRO_FEATURES="largefile opengl ptest multiarch
    wayland vulkan"
```

Créer son image `./recipes-image/rapsberrypi/myrpi.bb`

```
require recipes-graphics/images/core-image-weston.bb
IMAGE_FEATURES += "
    ssh-server-openssh \
"
IMAGE_INSTALL += " \
    setkey \
"
```

`ssh-server-openssh` permet d'avoir accès à la carte en ssh

`setkey` nouvelle recette permettant de passer qwerty en bépo

Testons notre image !

devtool

devtool

Exemple de l'utilisation de l'outil devtool

`devtool` est un outils très utiles lorsque l'on souhaite créer, développer ou modifier une recette et ses sources.

Les commandes de base :

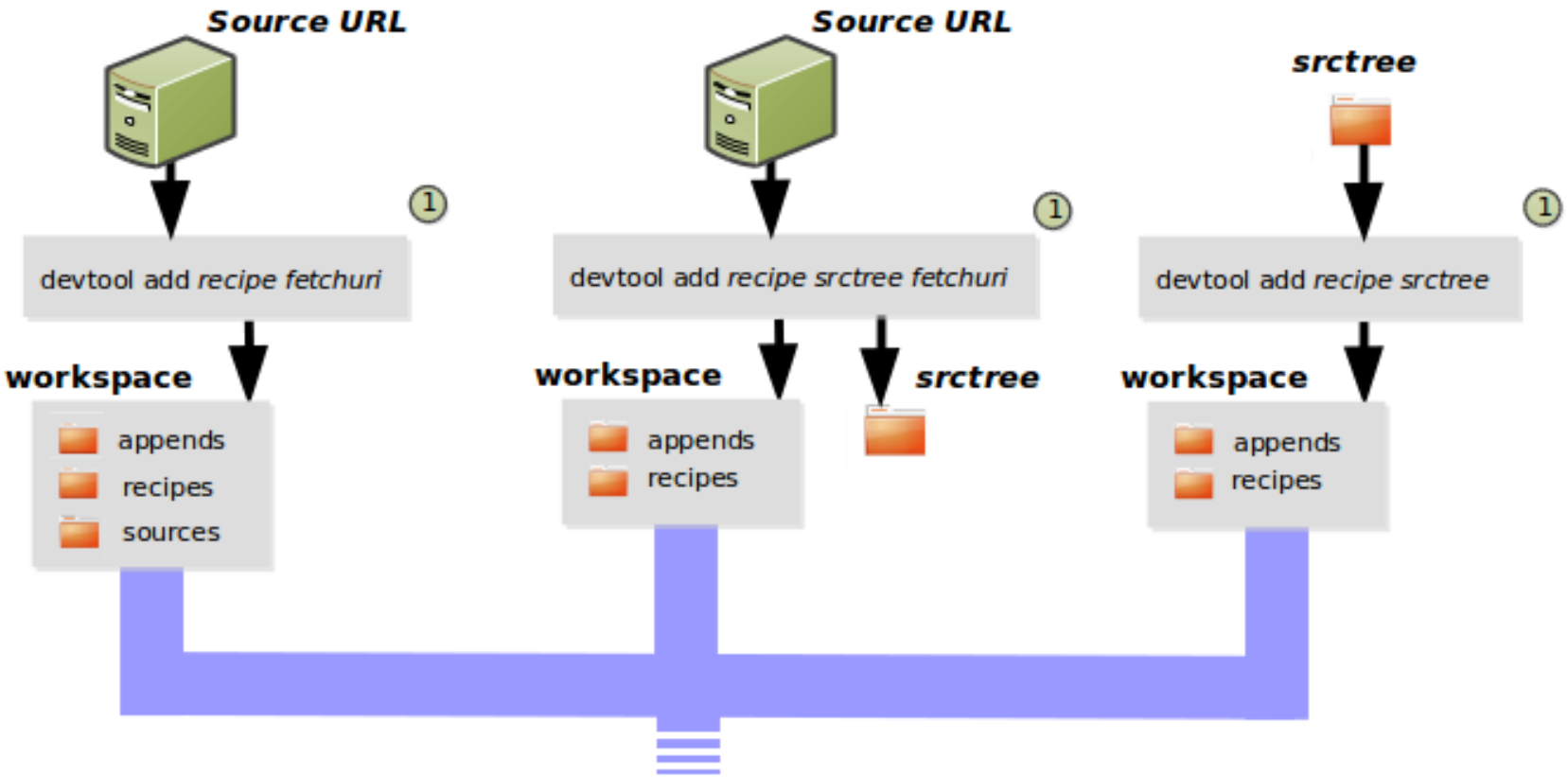
`devtool add` Ajoute un nouveau software à construire

`devtool modify` Génère un environnement pour modifier les sources d'un composant

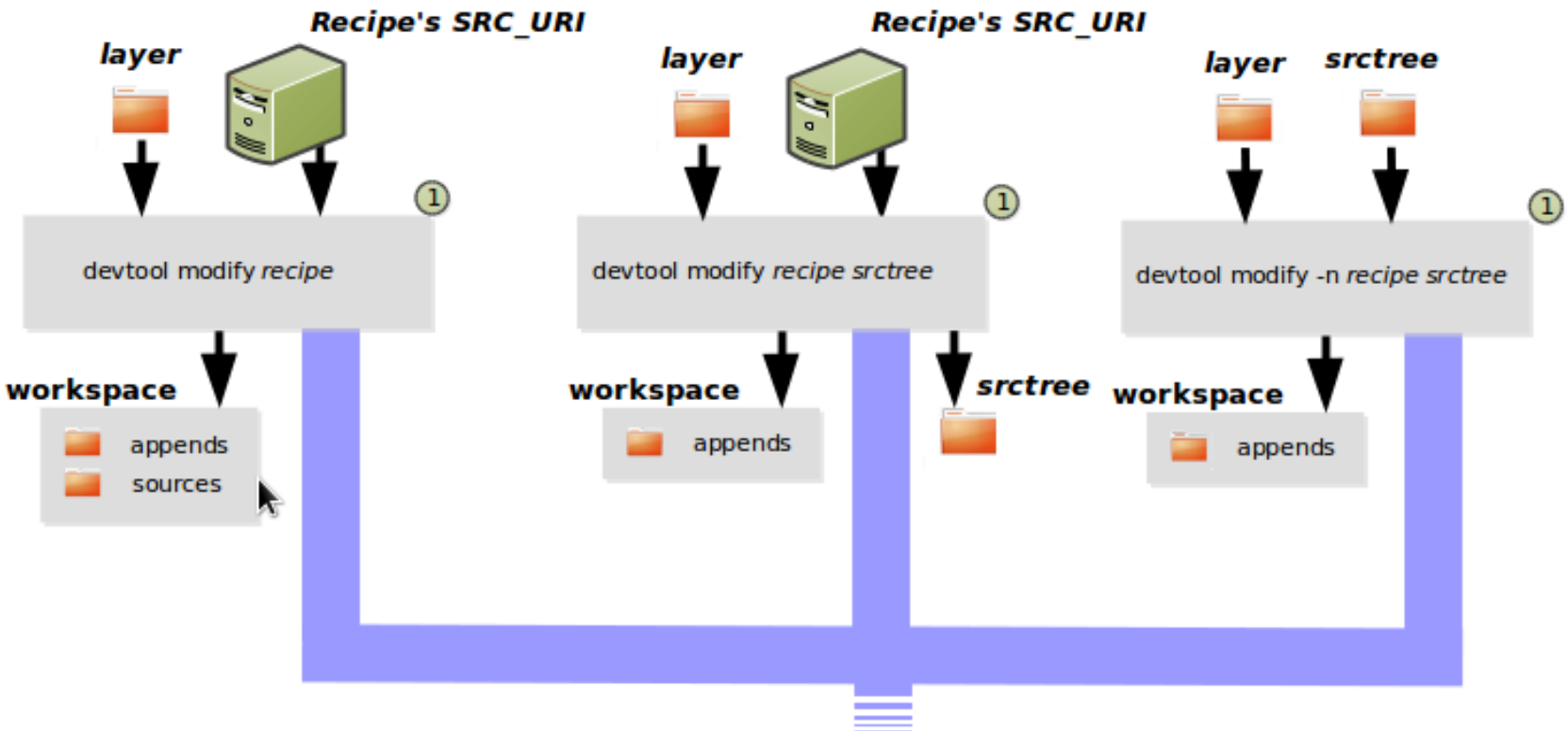
`devtool upgrade` Met à jour une recette existante

`devtool reset` Arrête le contexte de développement

Les sources peuvent provenir de plusieurs endroits différent



De même pour modifier une recette



Dès lors **devtool** créer un layer spécial (**workspace**) qui prend la priorité maximal sur les autres layers.

```
$ bitbake-layers show-layers
NOTE: Starting bitbake server...
layer             path                                      priority
==========================================================================
meta              /home/[...]/meta                          5
meta-poky         /home/[...]/meta-poky                     5
meta-yocto-bsp    /home/[...]/meta-yocto-bsp                5
workspace         /home/[...]/build/workspace               99
meta-raspberrypi  /home/[...]/build/../meta-raspberrypi     9
meta-python       /home/[...]/build/../meta-openembedded/meta-python 7
meta-oe           /home/[...]/build/../meta-openembedded/meta-oe 6
meta-meetup       /home/[...]/build/../meta-meetup          10
```

Dans ce layer on retrouve

- les sources mis sous git et patché
- un bbappend de la recette

```
$ cd $POKY/build/workspace/
$ tree -L 2
.
├── appends
│   └── weston_2.0.0.bbappend
├── conf
│   └── layer.conf
├── README
└── sources
    └── weston
```

Exemple pratique avec la recette `weston`

Modification avec `devtool` des sources de weston

```
$ devtool modify weston
$ cd $POKY/build/workspace/sources/weston/
$ vim libweston/compositor-wayland.c +1655
```

Ajout du patch "Fix an uninitialized variable"

```
@@ -1652,6 +1652,7 @@ input_handle_axis(void *data, struct
     wl_pointer *pointer,

     weston_event.axis = axis;
     weston_event.value = wl_fixed_to_double(value);
+    weston_event.has_discrete = false;

     if (axis == WL_POINTER_AXIS_VERTICAL_SCROLL &&
         input->vert.has_discrete) {
```

Les étapes :

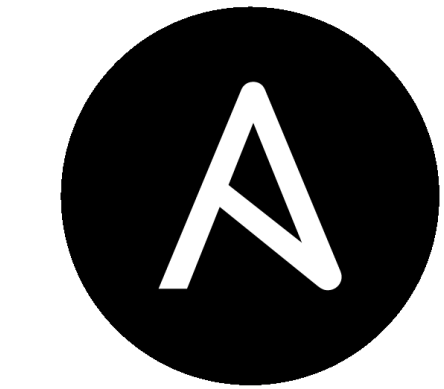
1. Faire la modification
2. Tester
3. Commiter
4. Appliquer la modification sous forme de patch

```
$ devtool update-recipe weston
[...]
NOTE: Adding new patch 0001-Fix-an-uninitialized-variable.
    patch
NOTE: Updating recipe weston_2.0.0.bb
```

5. Ajouter la modification dans son layer
6. Arrêter devtool

```
$ devtool reset weston
```

Ansible



Utiliser Ansible pour mettre en place un environnement Yocto

Definition

Ansible est un logiciel destiné à la configuration et la gestion de parc informatique.

Il permet de :

- déployer des logiciels
- gérer des configurations
- lancer des tâches

Pour :

- une machine donnée
- plusieurs machines

Un des avantage est qu'il utilise des fichiers de configuration au format YAML.

Le format YAML est :

- humainement lisibles
- plus facile à géré que certain autres formats

Défénition des cibles dans `/etc/ansible/hosts`

```
192.0.2.50
linuxembedded.exemple.fr
```

Pour lancer un programme à distance on peux soit spécifier

- tout les hôtes
- un hôte particulier

```
$ ansible all -a "/bin/ping 8.8.8.8 -c1"
$ ansible linuxembedded.exemple.fr -a "/bin/ping 8.8.8.8 -c1"
```

Ansible fournit un ensemble de modules qui permettent de lancer des actions spécifiques à distance.

```
$ ansible all -m ping
<address_ip> | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
```

Attention au faux amis, ici ping se connecte à un hôte, teste l'utilisabilité de python puis de renvoie le résultat pong en cas de succès

Les actions multiples – le principe de playbook

Pour effectuer plusieurs actions en une seule commande on utilise un playbook

```
$ ansible-playbook mon-fichier.yml
```

La syntaxe YAML

YAML

Le standard YAML a été créé en 2001 et est utilisé dans divers projets.

Un fichier YAML est formé de :

- variables
- dictionnaires (clé/valeur)
- listes

Les variables

Déclaration

```
vars:
  base_path: /mon/path
```

Accès à la varible

```
{{ base_path }}
```

Il est conseillé d'entourer la varible de guillemets

```
app_path: "{{ base_path }}/app"
```

Passer les variables en ligne de commande avec l'option

```
--extra-vars "base_path=/mon/path/"
```

Les dictionnaires

Les dictionnaires sont définis sous la forme *clé: valeur*.

```
# Information sur une personne
 martin:
   nom: Martin Devloper
   travail: Developer
   niveau: Experimente
```

Les listes

Les listes sont définies avec `-␣`, un tiret suivi d'un espace.

```
# Une liste de fruits
 - Pomme
 - Orange
 - Framboise
 - Mangue
```

Mélanger les syntaxes

D'autres choses plus complexes sont possibles en mélangeant les différentes syntaxes :

```
# Liste de plusieurs employes
- martin:
  nom: Martin D'vloper
  travail: Developer
  competences:
    - python
    - perl

- tabitha:
  nom: Tabitha Bitumen
  travail: Developer
  competences:
    - lisp
    - fortran
```

Les modules

Les modules lut par Ansible sont déclaré sous la forme *clé: valeur*.
Voici une liste non exhaustive de types de modules disponibles :

- git
- patch
- get_url
- shell
- copy
- service
- apt
- yum
- lxc_container
- make

Exemple

Le module `git` utilise de sous-options

```
# Exemple d'un telechargement de source git
- git:
  repo: 'https://git.yoctoproject.org/git/poky'
  version: krogoth
  dest: /home/user/poky
```

D'autres sous-options :

- `update: yes`
- `archive: /path/to/archive.zip`
- ...

Module particulier : hosts

Ce module est obligatoire.

Il fait référence aux hôtes dans `/etc/ansible/hosts`.

```
- hosts: all
  remote_user: root
```

Comment utilisé Ansible pour déployer un environnement de développement pour Yocto ?

Définir l'hôte

On utilise le `hosts` local

```
- hosts: 127.0.0.1
  connection: local
```

Répertoire de travail

Pour centraliser les sources on utilise un variable `TOP_SRCDIR`.
Elle sera passé en paramètre de la commande Ansible.

```
--extra-vars "TOP_SRCDIR=<path_to_top_srcdir>"
```

Ce qui donnera :

```
- name: "Get poky"
  git:
    repo: https://git.yoctoproject.org/git/poky
    version: krogoth
    dest: "{{TOP_SRCDIR}}"
```

```
- name: "Mon Projet"
  hosts: 127.0.0.1
  connection: local
  tasks:

  - name: "Get poky"
    git:
      repo: https://git.yoctoproject.org/git/poky
      version: krogoth
      dest: "{{TOP_SRCDIR}}"
      update: no

  - name: "Patch poky krogoth"
    patch:
      src: ../0001-qemu-Add-space-to-fix-concatenated-SRC_URI.patch
      basedir: "{{TOP_SRCDIR}}"
      strip: 1

  - name: "Copy hooks into poky"
    copy:
      src: /path/to/hooks/commit-msg
      dest: "{{TOP_SRCDIR}}/.git/hooks/commit-msg"
      mode: 755

  - name: "Get toolchain"
    get_url:
      url: http://192.168.2.200/src/toolchain/toolchain-M2.1.tgz
      checksum: md5:31a70bd7f7b74724af915a88fbe64f3d
      dest: "{{TOP_SRCDIR}}"

  - name: "Untar toolchain"
    shell: tar xzf {{TOP_SRCDIR}}/toolchain-M2.1.tgz -C {{TOP_SRCDIR}}/sdk/prebuilts/gcc/
```

Lancer

Pour lancer le projet :

```
$ ansible-playbook mon-fichier.yml --extra-vars "TOP_SRCDIR=/home
    /user/mon-projet/"
```

```
$ tree poky
poky/
.
├── bitbake
├── build
├── documentation
├── LICENSE
├── meta
├── meta-poky
├── meta-selftest
├── meta-skeleton
├── meta-yocto
├── meta-yocto-bsp
├── oe-init-build-env
├── oe-init-build-env-memres
├── README
├── README.hardware
├── sdk
├── scripts
└── toolchain-M2.1.tgz
```

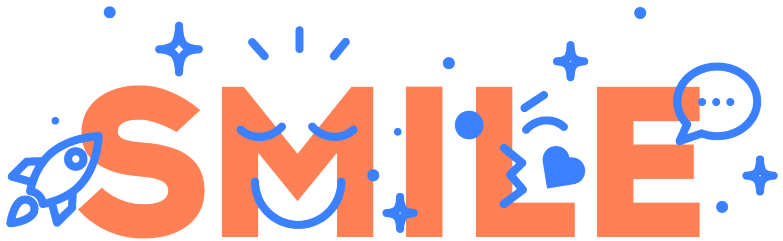
Sources

Ce document à été rédigé à partir des sources suivantes :

- `www.yoctoproject.org`
- `www.linuxembedded.fr`
- `www.ansible.com`
- `http://fabienlahouderepro.blogspot.fr/2017/03/building-weston-image-with-yocto-for.html`

Merci de votre attention !

Quetions ?



Mickaël Tansorier
mickael.tansorier@smile.fr
mickael@tansorier.fr