



Yocto Project® Kernel Lab, Hands-On

Trevor Woerner, Togán Labs

Yocto Project DevDay *Virtual*, North America, 2020



Ignoring TheYocto Project (briefly)

Kernel Workflow - first pass

- obtain kernel
- obtain cross-toolchain
- setup cross-toolchain/environment
- configure kernel/tweak DT
- build kernel/modules/dtb
- install kernel/modules/dtb
- test

Kernel Workflow - later

- edit kernel
- write kernel module
- do -rt stuff
- build, install, test

Obtain Kernel

- upstream Linus
- upstream linux-stable
- vendor kernel (tries to be good)
- evil vendor kernel (franken-kernel)

Obtain Cross-Toolchain

- cross-compiler from your distro
- gnu.gcc.org (compile your own, good luck)
- crosstool-NG (compile your own, with help)
- ARM/Linaro
- Bootlin
- kernel.org (NOTE: kernel only)
- vendor

Obtain Cross-Toolchain ARM/Linaro

arm Developer

IP PRODUCTS TOOLS AND SOFTWARE ARCHITECTURES SOLUTIONS COMMUNITY SUPPORT DOCUMENTATI

[Home](#) | [Tools and Software](#) | [Open Source Software](#) | [Developer Tools](#) | [GNU Toolchain](#) | [GNU Toolchain for the A-profile Architecture](#) | [Downloads](#)

GNU-A Downloads

[Overview](#) [GNU-A](#) [GNU-RM](#) [Architecture Support](#) [Specifications](#) [Feedback](#)

Downloads

The GNU Toolchain for the Cortex-A Family is a ready-to-use, open source suite of tools for C, C++ and Assembly programming targeting processors from the Arm Cortex-A family and implementing the Arm A-profile architecture.

The toolchain includes the GNU Compiler (GCC) and is available free of charge directly for Windows and Linux operating systems. Follow the links on this page to download the correct version for your development environment.

See the downloaded package's Release Notes (linked from this page) for full installation instructions.

GNU Toolchain for the A-profile Architecture Version 9.2-2019.12

Released: December 19, 2019

— GNU Toolchain for the A-profile Architecture: 9.2-2019.12 [December 19, 2019](#)

What's new in 9.2-2019.12

We are pleased to announce the Arm release of the pre-built GNU cross-toolchain for the A-profile cores: [GCC 9.2-2019.12](#)

Obtain Cross-Toolchain Bootlin - <https://toolchains.bootlin.com>

toolchains.bootlin.com

About

Toolchains

News

FAQ

About

This site provides a large number of ready-to-use cross-compilation toolchains, targeting the Linux operating system on a large number of architectures.

Based on [gcc](#) and [binutils](#), those toolchains are provided in several variants with the [glibc](#), [uClibc-ng](#) and [musl](#) C libraries. The toolchains are built using the [Bulldozer](#) build system.

Most toolchains are tested by building a Linux kernel and Linux userspace, and booting it under Qemu. This is of course not possible on some CPU architectures.

For each architecture and C library combination, two versions of the toolchain are provided:

- *stable*, which uses older proven versions of the toolchain components
- *bleeding-edge*, which uses the latest versions of the toolchain components

Useful resources:

- [License information for the toolchain components](#)
- [Source code for the toolchain components](#)
- [Report issues](#)
- [Tool used to build the toolchains](#)
- [This webpage](#)

Questions: checkout the [FAQ](#), or [contact us!](#)

Download

Select arch

aarch64

Select libc

glibc



Download stable



Download bleeding-edge

✔ Tests passed

[checksum \(sha256\)](#)

binutils	2.32
gcc	8.4.0
gdb	8.2.1
glibc	2.30-56-gc56022...
linux-headers	4.4.215

✔ Tests passed

[checksum \(sha256\)](#)

binutils	2.33.1
gcc	9.3.0
gdb	8.3
glibc	2.30-56-gc56022...
linux-headers	4.19.107

[View all aarch64 toolchains](#)

Obtain Cross-Toolchain

kernel.org -

<https://www.kernel.org/compilers>

Target Architecture		Host Architecture		
Kernel Arch	Compiler Arch	Intel 64-bit	ARM 64-bit	PowerPC 64-bit
Fully supported architectures				
arm64	aarch64-linux	10.1.0	10.1.0	10.1.0
alpha	alpha-linux	10.1.0	10.1.0	10.1.0
arm	arm-linux-gnueabi	10.1.0	10.1.0	10.1.0
arc	arc-linux	10.1.0	10.1.0	10.1.0
c6x	c6x-elf	10.1.0	10.1.0	10.1.0
csky	csky-linux	10.1.0	10.1.0	10.1.0
h8300	h8300-linux	10.1.0	10.1.0	10.1.0
parisc64	hppa64-linux	10.1.0	10.1.0	10.1.0
parisc	hppa-linux	10.1.0	10.1.0	10.1.0
ppc64le	ppc64le-linux	10.1.0	10.1.0	10.1.0

Setup Toolchain/Environment

- (compile +) install toolchain
- setup \$PATH
- either:
 - tweak environment

```
$ export ARCH=arm  
$ export CROSS_COMPILE=<toolchain-prefix>
```

- specify on cmdline

```
$ ARCH=arm CROSS_COMPILE=<toolchain-prefix> make
```

Setup Toolchain/Environment

Configure Kernel

```
[trevor resources]$ ls
linux-stable  toolchain
[trevor resources]$ cd linux-stable/
[trevor linux-stable (master u=)]$ make menucon
HOSTCC  scripts/basic/fixdep
UPD     scripts/kconfig/mconf-cfg
HOSTCC  scripts/kconfig/mconf.o
HOSTCC  scripts/kconfig/lxdialog/checklist.o
HOSTCC  scripts/kconfig/lxdialog/inputbox.o
HOSTCC  scripts/kconfig/lxdialog/menubox.o
HOSTCC  scripts/kconfig/lxdialog/textbox.o
HOSTCC  scripts/kconfig/lxdialog/util.o
HOSTCC  scripts/kconfig/lxdialog/yesno.o
HOSTCC  scripts/kconfig/confdata.o
```

<https://asciinema.org/a/34421>

6

Build Kernel, Modules, DTB

```
$ ARCH=arm CROSS_COMPILE=<toolchain-prefix> make
```

Install Kernel, Modules, DTB

- by default:
 - kernel: `$INSTALL_PATH` (default: `/boot`)
 - modules: `$INSTALL_MOD_PATH/lib/modules/$VER`
 - dtb: `$INSTALL_PATH/dtbs/$VER`
- this doesn't work for embedded
 - each board is... "unique"

Install Kernel, Modules, DTB

- how and where these various parts are installed is highly board-specific:
 - `extlinux.conf`
 - `uEnv.txt`
 - separate partitions
 - APPENDED_DTB (i.e. "bundled")
 - U-Boot FIT image

Install Kernel, Modules, DTB

- how and where these various parts are installed is highly board-specific:
 - "magic" offsets on storage device
 - multi-partition UBI (raw flash)
 - initial FAT/VFAT partition

Install Kernel, Modules, DTB

- there is a very tight coupling between
 - bootloader
 - kernel cmdline
 - kernel + modules + dtb
 - final image assembly (e.g. wic)

Edit Kernel

- not too difficult, locate and edit kernel sources
- just make sure you're editing the correct sources (multiple boards, multiple projects, multiple configurations) and using the correct toolchain (correctly)

Write Kernel Module

- need to copy+paste an example Makefile to get the out-of-kernel-tree build plumbing correct
- otherwise the mechanics aren't too complicated

Do -rt Stuff

- I hope the kernel version you're using is one for which there is an -rt patch available, otherwise forget it
- if the above is true, manually applying the -rt patch against an *unmodified* base isn't too difficult

Kernel Workflow Without Yocto

- it's not too hard if you've done it once or twice before
- but can be tricky to get it right the first time
- the amount of work compounds with the number of boards you need to support
 - multiple toolchains
 - multiple configurations
 - multiple install layouts

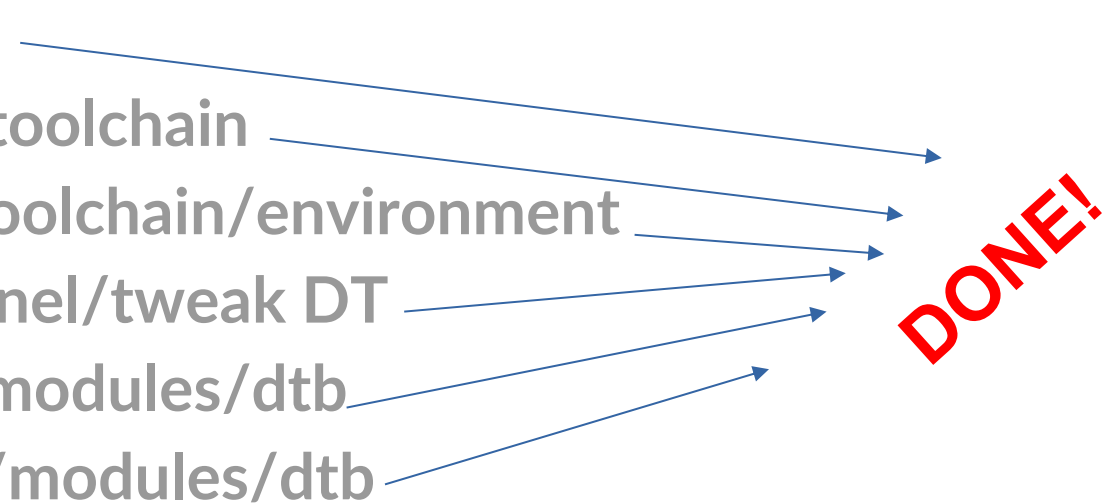


Can The Yocto Project Help?

Kernel Workflow - multiple users

- tweaking a kernel/config or writing a kernel module using an existing BSP
- board bring-up (creating a BSP) for a new board

Kernel Workflow - existing BSP

- obtain kernel
 - obtain cross-toolchain
 - setup cross-toolchain/environment
 - configure kernel/tweak DT
 - build kernel/modules/dtb
 - install kernel/modules/dtb
- 
- The diagram illustrates a six-step kernel workflow. Each step is represented by a blue bullet point followed by a text label. From each label, a blue arrow points towards the right, converging on a large, bold, red text 'DONE!' that is rotated diagonally. The steps are: 'obtain kernel', 'obtain cross-toolchain', 'setup cross-toolchain/environment', 'configure kernel/tweak DT', 'build kernel/modules/dtb', and 'install kernel/modules/dtb'.

Kernel Workflow TODO - existing BSP

- **change kernel**
- obtain cross-toolchain
- setup cross-toolchain/environment
- **configure kernel/tweak DT**
- build kernel/modules/dtb
- install kernel/modules/dtb
- **edit kernel sources**
- **write a kernel module**

Kernel Workflow TODO - existing BSP

change kernel

- many kernels from which to choose (upstream, vendor, stable)
- The Yocto Project provides its own (linux-yocto) plus tooling for use with linux-yocto, or your own
- linux-yocto comes in a couple variants:
 - (base)
 - -dev
 - -tiny
 - -rt

Kernel Workflow TODO - existing BSP tweak config

- use your own defconfig
- use an in-kernel *_defconfig*
- generate one or more config "fragments" which are applied on top of the base configuration

Kernel Workflow TODO - existing BSP

edit kernel code

Kernel Workflow TODO - existing BSP

write a module

Kernel Workflow - new BSP

- obtain kernel
- obtain cross-toolchain
- setup cross-toolchain/environment
- configure kernel/tweak DT
- build kernel/modules/dtb
- install kernel/modules/dtb



DONE!

Kernel Workflow - new BSP

- obtain kernel
- obtain cross-toolchain
- setup cross-toolchain/environment
- configure kernel/tweak DT
- build kernel/modules/dtb
- install kernel/modules/dtb
- ... *but with a lot of help!*



DONE!

Kernel Workflow - new BSP

- obtain kernel
 - implicit understanding of how to fetch code
- setup cross-toolchain/environment
 - set the processor "TUNE" in \$MACHINE
- configure kernel/tweak DT
 - lots of linux-yocto tooling
- install kernel/modules/dtb
 - lots of kernel-handling classes



Hands On: Existing BSP

Hands-On: Prep

```
ilab01:~$ cd scratch
ilab01:~/scratch$ git clone -b dunfell git://git.yoctoproject.org/poky.git
Cloning into 'poky'...
remote: Enumerating objects: 479650, done.
remote: Counting objects: 100% (479650/479650), done.
remote: Compressing objects: 100% (113534/113534), done.
remote: Total 479650 (delta 359015), reused 478925 (delta 358472)
Receiving objects: 100% (479650/479650), 164.40 MiB | 28.37 MiB/s, done.
Resolving deltas: 100% (359015/359015), done.
```

Hands-On: Prep

```
ilab01:~/scratch$ . poky/oe-init-build-env
...
### Shell environment set up for builds. ###
...
ilab01:~/scratch/build$
```

- I like to remind myself this is now a modified-environment shell for builds:

```
ilab01:~/scratch/build$ export $PS1="${PS1}lab> "
ilab01:~/scratch/build$ lab>
```

Hands-On: Prep

```
ilab01:~/scratch/build$ lab> $EDITOR conf/local
```

- here's what you need to change:

```
MACHINE = "qemux86-64"  
DL_DIR = "/scratch/downloads"  
SSTATE_DIR = "/scratch/sstate-  
cache"  
PACKAGE_CLASSES ?= "package_ipk"  
SDKMACHINE = "x86_64"  
OE_TERMINAL = "screen"
```

Hands-On: Prep

- **baseline build:**

```
ilab01:~/scratch/build$ lab> bitbake core-image-minimal
Build Configuration:
BB_VERSION           = "1.46.0"
BUILD_SYS            = "x86_64-linux"
NATIVELSBSTRING     = "universal"
TARGET_SYS           = "x86_64-poky-linux"
MACHINE              = "qemux86-64"
DISTRO               = "poky"
DISTRO_VERSION       = "3.1.1"
TUNE_FEATURES        = "m64 core2"
TARGET_FPU           = ""
meta
meta-poky
meta-yocto-bsp       = "dunfell:93ef4736915090ac9a2402916df8924ac4439490"
```

Hands-On: Prep

- verify it runs

```
ilab01:~/scratch/build$ lab> runqemu slirp nographic serial
...
hwclock: settimeofday: Invalid argument
chmod: /var/log/wtmp: No such file or directory
Failed to set mode -0664- for -/var/log/wtmp-.
INIT: Entering runlevel: 5
Configuring network interfaces... ip: RTNETLINK answers: File exists
hwclock: settimeofday: Invalid argument
Starting syslogd/klogd: done

Poky (Yocto Project Reference Distro) 3.1.1 qemux86-64 /dev/ttyS0

qemux86-64 login:
```

Hands-On: Prep

- shutdown

```
Poky (Yocto Project Reference Distro) 3.1.1 qemu86-64 /dev/ttyS0

qemu86-64 login: root
root@qemu86-64:~# uname -a
Linux qemu86-64 5.4.43-yocto-standard #1 SMP PREEMPT Thu May 28 15:33:30 UTC
2020
root@qemu86:~# shutdown -h now
...
[ 130.239174] ACPI: Preparing to enter system sleep state S5
[ 130.242375] reboot: Power down
runqemu - INFO - Cleaning up
ilab01:~/scratch/build$ lab>
```

Hands-On: Prep

- create/add a layer for your changes

```
ilab01:~/scratch/build$ lab> bitbake-layers create-layer meta-mytweaks
NOTE: Starting bitbake server...
Add your new layer with 'bitbake-layers add-layer meta-mytweaks'
ilab01:~/scratch/build$ lab> bitbake-layers add-layer meta-mytweaks
NOTE: Starting bitbake server...
ilab01:~/scratch/build$ lab>
```

Hands-On: Edit the Kernel

Hands-On: Edit the Kernel

- use *devtool*

```
ilab01:~/scratch/build$ lab> devtool modify virtual/kernel
...
Currently 1 running tasks (431 of 437) 98% | ##### |
0: linux-yocto-5.4.43+gitAUTOINC+aafb8f095e_9e1b13d7f9-r0 do_kernel_checkout
- 1m49s (pid 1615575)
INFO: Copying kernel config to srctree
INFO: Source tree extracted to /scratch/build/workspace/sources/linux-yocto
INFO: Recipe linux-yocto now set up to build from
/scratch/build/workspace/sources/linux-yocto
```

Hands-On: Edit the Kernel

- edit init/main.c

```
ilab01:~/scratch/build$ lab> $EDITOR workspace/sources/linux-yocto/init/main.c
1137      /*
1138      * We try each of these until one succeeds.
1139      *
1140      * The Bourne shell can be used instead of init if we are
1141      * trying to recover a really broken machine.
1142      */
1143      pr_notice("*** --- >>> edit <<< --- ***\n");
1144      if (execute_command) {
1145          ret = run_init_process(execute_command);
1146          if (!ret)
1147              return 0;
1148          panic("Requested init %s failed (error %d).",
1149              execute_command, ret);
```

Hands-On: Edit the Kernel

- **build**

```
ilab01:~/scratch/build$ lab> bitbake core-image-minimal
...
NOTE: linux-yocto: compiling from external source tree
/scratch/build/workspace/sources/linux-yocto
...
```

Hands-On: Edit the Kernel

- test (you might have to scroll up)

```
ilab01:~/scratch/build$ lab> runqemu slirp nographic serial
...
[ 5.209719] VFS: Mounted root (ext4 filesystem) on device 253:0.
[ 5.211596] devtmpfs: mounted
[ 5.312611] Freeing unused kernel image memory: 1580K
[ 5.313286] Write protecting kernel text and read-only data: 20480k
[ 5.313744] *** --- >>> edit <<< --- ***
[ 5.313980] Run /sbin/init as init process
INIT: version booting
Starting udev
[ 6.096177] udevd[138]: starting version 3.2.9
[ 6.162295] udevd[139]: starting eudev-3.2.9
...
```

Hands-On: Edit the Kernel

- commit your changes

```
ilab01:~/scratch/build$ lab> pushd workspace/sources/linux-yocto
~/scratch/build/workspace/sources/linux-yocto ~/scratch/build
ilab01:~/scratch/build$ lab> git commit -avs -m "init/main.c: add my tweaks"
ilab01:~/scratch/build$ lab> popd
~/scratch/build
```

Hands-On: Edit the Kernel

- update your metadata

```
ilab01:~/scratch/build$ lab> devtool finish linux-yocto meta-mytweaks
...
NOTE: Writing append file /scratch/build/meta-mytweaks/recipes-
kernel/linux/linux-yocto_%.bbappend
NOTE: Copying 0001-init-main.c-add-my-tweaks.patch to /scratch/build/meta-
mytweaks/recipes-kernel/linux/linux-yocto/0001-init-main.c-add-my-
tweaks.patch
INFO: Cleaning sysroot for recipe linux-yocto...
INFO: Leaving source tree /scratch/build/workspace/sources/linux-yocto as-is;
if you no longer need it then please delete it manually
ilab01:~/scratch/build$ lab> rm -fr workspace/sources/linux-yocto
```

Hands-On: Edit the Kernel

- check

```
ilab01:~/scratch/build$ lab> tree meta-mytweaks
meta-mytweaks/
├── COPYING.MIT
├── README
├── conf
│   └── layer.conf
├── recipes-kernel
│   └── linux
│       ├── linux-yocto
│       │   └── 0001-init-main.c-add-my-tweaks.patch
│       └── linux-yocto_%.bbappend
```

Hands-On: Tweak the Kernel Configuration

Hands-On: Tweak the Kernel Configuration

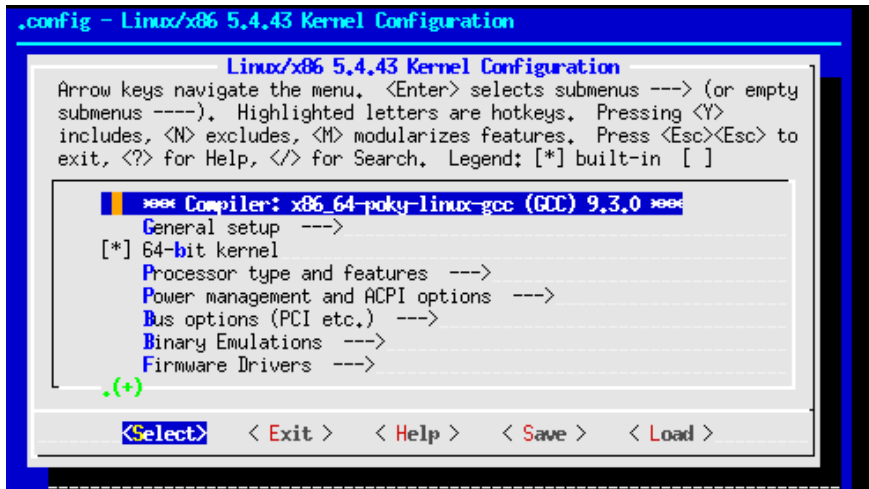
- use devtool, setup

```
ilab01:~/scratch/build$ lab> devtool modify virtual/kernel
...
NOTE: Tasks Summary: Attempted 437 tasks of which 426 didn't need to be rerun
and all succeeded.
INFO: Copying kernel config to srctree
INFO: Source tree extracted to /z/3.1-build-dunfell/kernellab/x86-
64/build/workspace/sources/linux-yocto
INFO: Recipe linux-yocto now set up to build from /z/3.1-build-
dunfell/kernellab/x86-64/build/workspace/sources/linux-yocto
```

Hands-On: Tweak the Kernel Configuration

- use devtool

```
ilab01:~/scratch/build$ lab> devtool menuconfig linux-yocto
```



The screenshot shows the 'Linux/x86 5.4.43 Kernel Configuration' menuconfig interface. At the top, it displays the title and a detailed instruction block: 'Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in []'. Below this is a scrollable list of configuration options. The first option is 'Compiler: x86_64-poky-linux-gcc (GCC) 9.3.0', which is currently selected and highlighted in blue. Other visible options include 'General setup --->', '[*] 64-bit kernel', 'Processor type and features --->', 'Power management and ACPI options --->', 'Bus options (PCI etc.) --->', 'Binary Emulations --->', and 'Firmware Drivers --->'. At the bottom of the window, there is a navigation bar with buttons for '<Select>', '<Exit >', '<Help >', '<Save >', and '<Load >'. A green '(+)' symbol is visible in the bottom left corner of the menu area.

Hands-On: Tweak the Kernel Configuration

- navigate to:
 - General Setup
 - Preemption Model
- select:
 - No Forced Preemption (Server)
- hit double-esc or <Exit> and save your changes

```
INFO: Updating config fragment /z/3.1-build-dunfell/kernellab/x86-64/build/workspace/sources/linux-yocto/oe-local-files/devtool-fragment.cfg
```

Hands-On: Tweak the Kernel Configuration

- test

```
ilab01:~/scratch/build$ lab> bitbake core-image-minimal
NOTE: linux-yocto: compiling from external source tree
/scratch/build/workspace/sources/linux-yocto
ilab01:~/scratch/build$ lab> runqemu slirp nographic serial
...
[ 0.223909] rcu: Hierarchical RCU implementation.
...
qemux86-64 login: root
root@qemux86-64:~# uname -a
Linux qemux86-64 5.4.43-yocto-standard #1 SMP Wed Jul 1 20:06:32 UTC 2020
x86_64 GNU/Linux
```

- **NOTE:** missing "Preemptible hierarchical RCU" and "PREEMPT" in "uname -a"

Hands-On: Tweak the Kernel Configuration

- update metadata

```
ilab01:~/scratch/build$ lab> devtool finish linux-yocto meta-mytweaks
...
NOTE: Writing append file /scratch/build/meta-mytweaks/recipes-
kernel/linux/linux-yocto_%.bbappend
NOTE: Copying devtool-fragment.cfg to /scratch/build/meta-mytweaks/recipes-
kernel/linux/linux-yocto/devtool-fragment.cfg
INFO: Cleaning sysroot for recipe linux-yocto...
INFO: Leaving source tree /scratch/build/workspace/sources/linux-yocto as-is;
if you no longer need it then please delete it manually
ilab01:~/scratch/build$ lab> rm -fr workspace/sources/linux-yocto
```

Hands-On: Tweak the Kernel Configuration

- check

```
ilab01:~/scratch/build$ lab> tree meta-mytwaks
meta-mytwaks/
├── COPYING.MIT
├── README
├── conf
│   └── layer.conf
├── recipes-kernel
│   └── linux
│       ├── linux-yocto
│       │   ├── 0001-init-main.c-add-my-tweaks.patch
│       │   └── devtool-fragment.cfg
│       └── linux-yocto_%.bbappend
```

Hands-On: Tweak the Kernel Configuration

- check

```
ilab01:~/scratch/build$ lab> cat meta-mytweaks/recipes-kernel/linux/linux-  
yocto/devtool-fragment.cfg  
CONFIG_PREEMPT_NONE=y  
# CONFIG_PREEMPT is not set  
CONFIG_TREE_RCU=y  
CONFIG_INLINE_SPIN_UNLOCK_IRQ=y  
CONFIG_INLINE_READ_UNLOCK=y  
CONFIG_INLINE_READ_UNLOCK_IRQ=y  
CONFIG_INLINE_WRITE_UNLOCK=y  
CONFIG_INLINE_WRITE_UNLOCK_IRQ=y
```



The Yocto Project Kernel Tooling



yocto
PROJECT

THE
LINUX
FOUNDATION