

Development environment setup

Carlos Maiolino

Izabela Bakollari

Rado Vrbosky

Vratislav Bende

Leonardo Vaz

Overview

- ▶ Development and testing machines
- ▶ Development tools
- ▶ Kernel configuration
- ▶ Installing the Linux kernel
- ▶ Patch formatting and submission



Development and testing machines

Why two different machines?

- ▶ If you screw up your code, you don't lose your dev environment
- ▶ You can use bare-metal for development if you have a Linux machine
- ▶ We recommend Fedora, but you can use any distro as long as you know how to use it.
- ▶ Why Fedora?
 - Bleeding edge tools available (we don't need to compile anything other than Linux itself).
 - We know how the package manager works

Development machine

- ▶ Can be your bare-metal machine if you use Linux
- ▶ If you don't use Linux, you will need to install a virtual machine
- ▶ Setup the development environment (more on this later)
- ▶ NFS server: Easy way to build the kernel in one place and install in another

Test machine

- ▶ A Linux machine where we will install and test the Linux kernel
- ▶ Don't need to be powerful
- ▶ At least 2 vCPUs would be great so we can use SMP
- ▶ As much memory as you have available
- ▶ NFS client (to install the kernel)



Development Tools

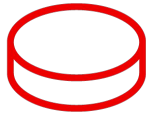
Some useful tools

- ▶ git (mandatory)
- ▶ Linux source tree (of course)
- ▶ Compiler (gcc, clang)
- ▶ code editor (vim, emacs, whatever else you want to use)
- ▶ Navigation tools
- ▶ Debug tools (To be discussed later)

Obtaining the source code

- ▶ Make sure to use the correct tree
 - You may need to submit your patch to a different tree than Linus's main tree
- ▶ Mainline, stable, subsystem trees
- ▶ We will use Linus's main tree for the purposes of the course

Linux Kernel Flavours



Maintainer Subtree

Receive and merge patches related to a specific subsystem or subcomponent



Linux-Next tree

Aggregate bleeding edge patches, to be tested before being merged to the main tree



Vanilla

Linus' main tree, the end point of all Linux's patches.



Quick look into kernel configuration

Configure your kernel

- ▶ How the kernel `.config` file works
- ▶ How to create the config file the easy way
 - Copy from a distribution and change it
 - Use kernel config generator
 - Graphical tools (xconfig, gconfig, nconfig, menuconfig)
- ▶ The hard way: `$make config`
- ▶ The spartan way: write the `.config` yourself!



Building and installing the kernel

Building

- ▶ In order to build the kernel, the `.config` should be ready
 - You can tweak the version if you want (see `localversion` file)
- ▶ Distribution package vs standard build vs Tarball
 - Run `$ make help` and look for the options
- ▶ Run `$ make -jX >/dev/null` to start building the kernel
 - Where **X** depends on how many CPUs you have available
- ▶ Wait a long time
- ▶ Hope for no errors (otherwise you'll need to start it over).

Installing

- ▶ Transferring the built kernel image to the test machine
 - Copying the package
 - Packaging the executables and copying them
 - Accessing the dev environment via NFS
- ▶ The development environment should be the same architecture
 - Unless you are cross-compiling a kernel for a different architecture
- ▶ Make sure your kernel is finally bootable



Browsing the Kernel Tree

Kernel Directory Structure

Documentation/
scripts/
tools/
MAINTAINERS
README

arch/
crypto/
include/
kernel/
lib/

block/
drivers/
fs/
mm/
net/
virt/

<https://makelinux.github.io/kernel/map/>



Environment examples

Carlos

- ▶ Git
 - git-worktree
 - guilt
- ▶ tmux
- ▶ vim + nerdtree + tagbar
- ▶ cscope
- ▶ neomutt

Izabella

- ▶ Git
 - git-worktree
 - guilt
- ▶ tmux
- ▶ vim + nerdtree + tagbar
- ▶ cscope
- ▶ neomutt

Rado

- ▶ git
- ▶ screen
- ▶ vim + nerdtree + tagbar
- ▶ make tags, grep
- ▶ mutt, gitlab
- ▶ qemu

Vrato

- ▶ git
- ▶ vim (quite raw TBH)
- ▶ cscope, gtags, grep
- ▶ perf, trace-cmd, systemtap
- ▶ bash & python scripts



Linux coding style and patch submission process

Linux Kernel coding style

- ▶ Linux maintainers are strict regarding coding style
- ▶ Make sure your code follows it
- ▶ There are tools for checking the code style
 - Coding style check script (`scripts/checkpatch.pl`)
 - vim plugin (if you use vim)
- ▶ Coding Style in the following URL:
<https://docs.kernel.org/process/coding-style.html>

Prepare your patch for submission

- ▶ Avoid heated discussions in the mailings
- ▶ Make sure that
 - Your patch applies against the tree you are submitting it
 - It builds
 - The kernel boots and it doesn't crash the system immediately
- ▶ Beginner friendly tool: `$ git format-patch`

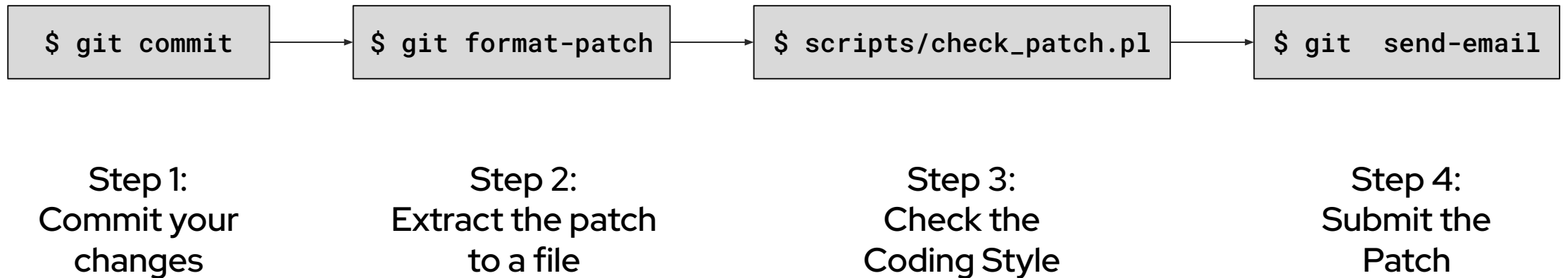
Where should we send the patch?

- ▶ Look for a mailing list related to what you are changing
 - Most of the time, patches are not submitted against the main tree
- ▶ Make sure your patch is tested on the right tree before submitting
- ▶ Use `scripts/get_maintainer.pl` to find the subsystem maintainer

Send it!

- ▶ Linux upstream community is email based
- ▶ You can use `git send-email`
- ▶ Configure your `~/ .gitconfig` to submit patches
- ▶ See the documentation for examples

Recap



Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.



[linkedin.com/company/red-hat](https://www.linkedin.com/company/red-hat)



[youtube.com/user/RedHatVideos](https://www.youtube.com/user/RedHatVideos)



[facebook.com/redhatinc](https://www.facebook.com/redhatinc)



twitter.com/RedHat