Open Source Software Development

Linux Kernel Development

Thái Minh Tuấn - Email: minhtuan@ctu.edu.vn

Learning Objectives

- Linux Kernel Development Process
- Release Cycle - Kernel Trees - Patches
- Working on Linux community
- Building and Installing Your First Kernel
- Writing Your First Kernel Patch
- Testing - Debugging
- Continue Your Kernel Journey

Linux Kernel Development Process

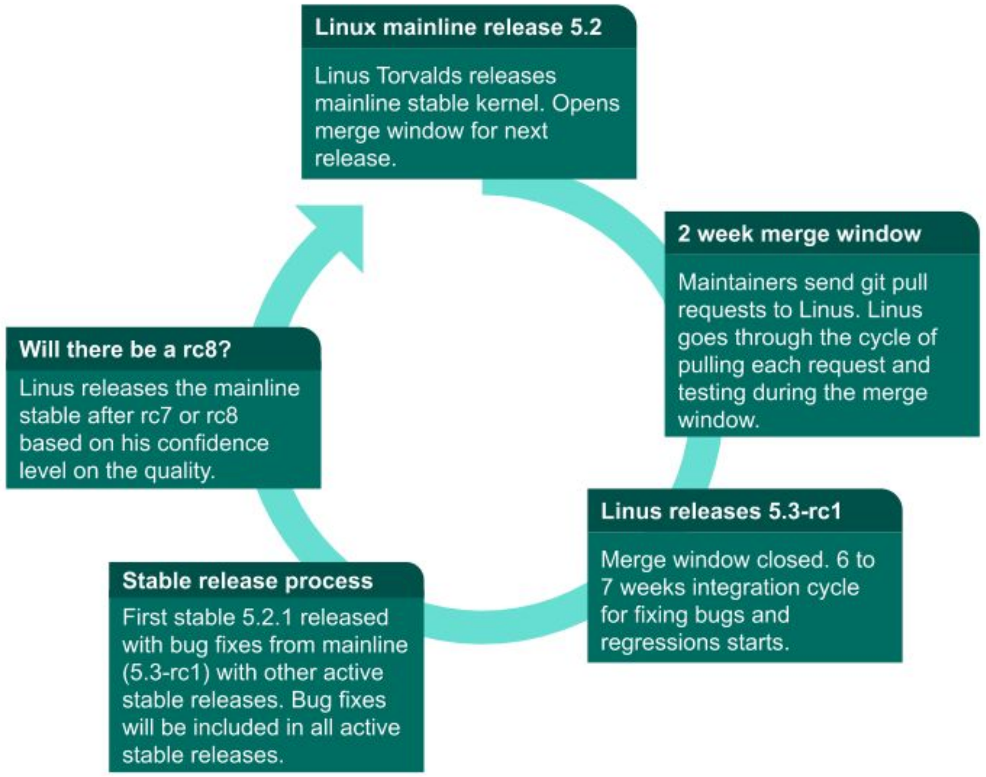
- The result of collaborative development efforts from developers across the globe
  - Over 13,000 kernel developers (2020)
- *Patches*: Small incremental changes
  - Add new features, make enhancements, and fix bugs
- *Releases*: New versions of Linux kernel
- Linux kernel development process:
  - 24-hour, 7 days a week, and 365 days of continuous process
  - New release once every 2+ months (10 to 11 weeks)
    - Several stable and extended stable releases once a week
  - Time-based rather than feature-based
    - Not held up for features, no set date for releases
  - New development and current release integration cycles run in parallel

Release Cycle (1/2)

- Linus Torvalds releases a new kernel and opens a *2-week merge window*
  - Pulls code for the next release from subsystem maintainers
  - Subsystem maintainers send signed git pull requests to Linus either during the merge window or before
  - All major new development is added to the kernel
  - 10,000+ change sets (patches) get pulled into Linus's tree
- At the end of merge window, Linus releases the first release candidate (rc1)
  - Release cycle moves into a bug fixes-only mode
  - Once a week, a new rc comes out: rc2, rc3, rc4, etc.
    - Until all major bug fixes and regressions (if any) are resolved

Release Cycle (2/2)

- New cycle begins with a *3-week quiet period*: Starts a week before the release, and continues through the 2-week merge window
    - Maintainers/contributors are busy getting their trees ready to send pull requests to Linus



Linux mainline release 5.2
Linus Torvalds releases mainline stable kernel. Opens merge window for next release.
2 week merge window
Maintainers send git pull requests to Linus. Linus goes through the cycle of pulling each request and testing during the merge window.
Linus releases 5.3-rc1
Merge window closed. 6 to 7 weeks integration cycle for fixing bugs and regressions starts.
Stable release process
First stable 5.2.1 released with bug fixes from mainline (5.3-rc1) with other active stable releases. Bug fixes will be included in all active stable releases.
Will there be a rc8?
Linus releases the mainline stable after rc7 or rc8 based on his confidence level on the quality.

Kernel Trees

- Mainline kernel tree
  - Maintained by Linus Torvalds
  - Releases mainline kernels and RC releases
- Stable tree
  - Maintained by Greg Kroah-Hartman
  - Consists of stable release branches
- Linux-next tree
  - The integration tree maintained by Stephen Rothwell
  - Code from a large number of subsystem trees gets pulled into this tree periodically and then released for integration testing

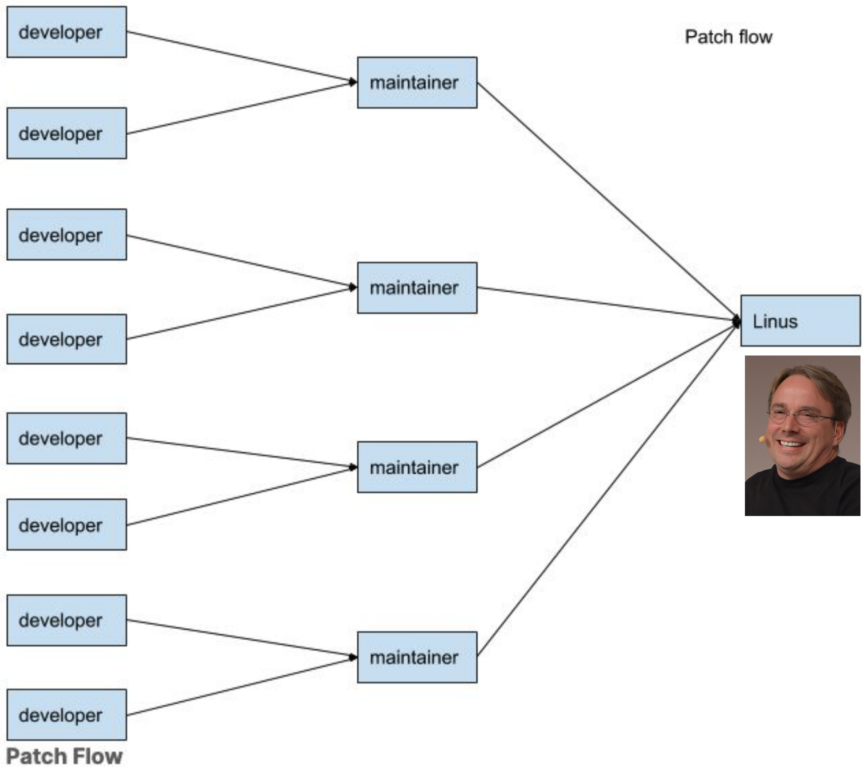
Active Kernel Releases

- Release Candidate (RC)
  - Mainline kernel pre-releases that are used for testing new features
  - Developers test these releases for bugs and regressions
- Stable
  - Bug fix-only releases. Released once a week, or on an as needed basis
  - After mainline kernel released, it moves into stable mode
  - Bug fixes for a stable kernel are backported from the mainline kernel
  - Only maintained for a few mainline release cycles
- Long-term
  - Stable releases selected for long-term maintenance
    - Provide critical bug fixes for older kernel trees
  - Maintained for a longer period
    - Allow multiple vendors collaborate on a specific kernel release
      - Plan on maintaining for an extended period of time

https://www.kernel.org/category/releases.html

Subsystem Maintainers

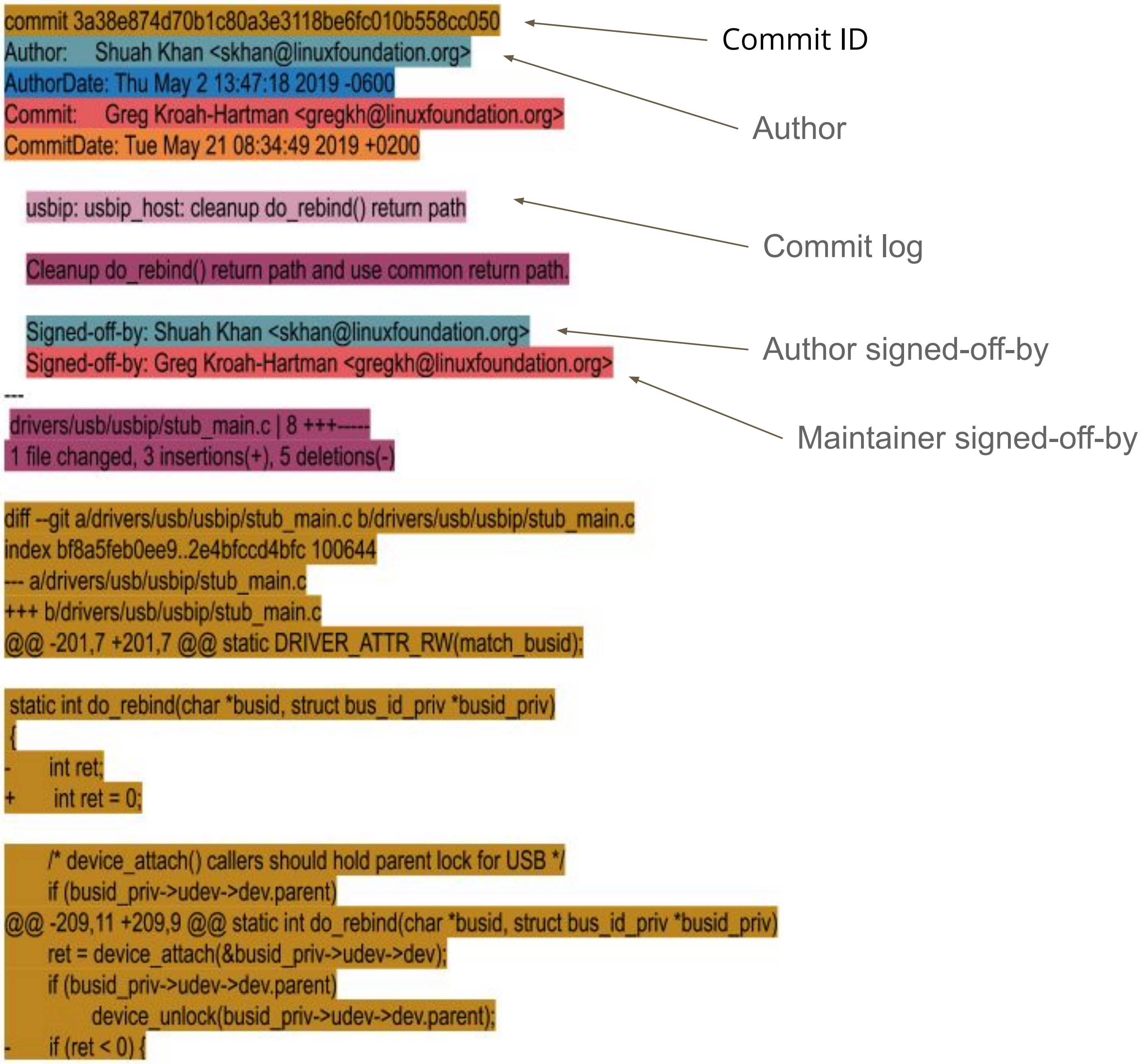
- Each major subsystem has its own tree and designated maintainer(s)
    - MAINTAINERS file: List of subsystems and their maintainers
- Development process itself happens entirely over emails
    - Almost every kernel subsystem has a mailing list
    - Contributors send patches to mailing lists through emails
    - Contributions are then discussed through emails
    - Maintainers send an email response to contributors when they accept and commit patches



"It is extremely amazing how the Linux community, which is worldwide, does all its work in email."

Patches

- Small incremental changes made to the kernel
    - Developers send to the kernel mailing lists through email
- Independent modification that stands on its own
    - Cannot break the kernel build
    - Easier to isolate regressions
    - Complex changes to the kernel are thus split into smaller chunks
    - Find an existing compile warning while making a code change
        - Fix it independently in a separate patch instead of combining it with your code change
- Maintainers have their personal preferences on how granular the patch splitting should be for their subsystems
    - Giving feedback on their preferences during the patch review



```
commit 3a38e874d70b1c80a3e3118be6fc010b558cc050
Author:     Shuah Khan <skhan@linuxfoundation.org>
AuthorDate: Thu May 2 13:47:18 2019 -0600
Commit:     Greg Kroah-Hartman <gregkh@linuxfoundation.org>
CommitDate: Tue May 21 08:34:49 2019 +0200

    usbip: usbip_host: cleanup do_rebind() return path

    Cleanup do_rebind() return path and use common return path.

    Signed-off-by: Shuah Khan <skhan@linuxfoundation.org>
    Signed-off-by: Greg Kroah-Hartman <gregkh@linuxfoundation.org>
---
 drivers/usb/usbip/stub_main.c | 8 +++-----
 1 file changed, 3 insertions(+), 5 deletions(-)

diff --git a/drivers/usb/usbip/stub_main.c b/drivers/usb/usbip/stub_main.c
index bf8a5feb0ee9..2e4bfccd4bfc 100644
--- a/drivers/usb/usbip/stub_main.c
+++ b/drivers/usb/usbip/stub_main.c
@@ -201,7 +201,7 @@ static DRIVER_ATTR_RW(match_busid);

 static int do_rebind(char *busid, struct bus_id_priv *busid_priv)
 {
-	int ret;
+	int ret = 0;

 	/* device_attach() callers should hold parent lock for USB */
 	if (busid_priv->udev->dev.parent)
@@ -209,11 +209,9 @@ static int do_rebind(char *busid, struct bus_id_priv *busid_priv)
 	ret = device_attach(&busid_priv->udev->dev);
 	if (busid_priv->udev->dev.parent)
 		device_unlock(busid_priv->udev->dev.parent);
-	if (ret < 0) {
```

Create and Apply Git Patch Files

- Modify the user-specific Git configuration file `~/.gitconfig`
- Create Git patch files
  - `git format-patch <branch> <options>`
  - Example: Create Git patch for 3 topmost commits
    - `git format-patch --pretty=fuller -3`
- Apply Git patch files
  - `git am <patch_file>`

Patch Tags

- Acked-by: Used by the maintainer of the affected code when that maintainer neither contributed to, nor forwarded the patch
- Reviewed-by: Indicates that the patch has been reviewed by the person named in the tag
- Reported-by: Gives credit to people who find bugs and report them.
- Tested-by: Indicates that the patch has been tested by the person named in the tag
- Suggested-by: Give credit for the patch idea to the person named in the tag
- Fixes: Indicates that the patch fixes an issue in a previous commit referenced by its Commit ID, allows us track where the bug originated

Patch Email Subject Line Conventions

- [PATCH] prefix is used to indicate that the email consists of a patch
  - [PATCH v4] is used to indicate that the patch is the 4th version of this specific change that is being submitted
- [PATCH RFC] or [RFC PATCH] indicates the author is requesting comments on the patch
  - RFC stands for "Request For Comments"
- NOT unusual for a patch to go through a few revisions before it gets accepted
  - An artifact of collaborative development
  - The goal is to get the code right and not rush it in

Patch Version History

- Include the patch version history when sending a re-worked patch
- The patch revision history on what changed between the current version and the previous version is added between the “---” and the “start of the diff” in the patch file
  - Any text that is added here gets thrown away and will not be included in the commit when it is merged into the source tree
  - It is good practice to include information that helps with reviews and doesn’t add value to the commit log
  - Check mailing lists to get a feel for what kind of information gets added
- Do not send new versions of a patch as a reply to a previous version
- Patch example with version history for the v2 version

Linux Kernel Contributor Covenant Code of Conduct

- Linux kernel community abides by the Linux Kernel Contributor Covenant Code of Conduct
    - Contributors and maintainers pledge to foster an open and welcoming community to participate in and become part of
- Linux kernel is often a labor of love of a community that truly thinks independently together
    - We value diversity of thought in solving technical problems
- How to conduct yourself as a contributor and participant in the community
    - Contributor Covenant Code of Conduct
    - Linux Kernel Contributor Covenant Code of Conduct Interpretation
- It is important to understand that when you send a patch with your sign-off, you are agreeing to abide by the code of conduct outlined in these two documents

Linux Enforcement Statement

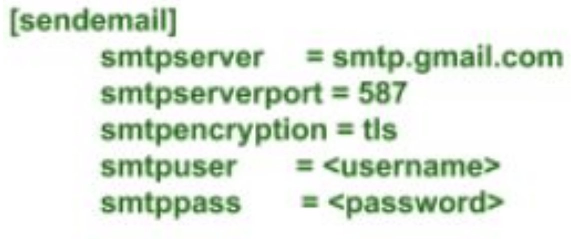
- Linux Kernel is provided under the terms of the GNU General Public License version 2 only (GPL-2.0)
    - With an explicit syscall exception described here
- Linux Kernel Enforcement Statement
    - How our (Linux Kernel) software is used and how the license for our software is enforced
    - Balance the enforcement actions with the health and growth of the Linux kernel ecosystem
- You are encouraged to read the statement to understand the intent and spirit of the statement and to get a better understanding of our community
    - *Working together, we will be stronger*

Configuring Your Development System (1/3)

- Getting Your System Ready
    - A regular laptop is a good choice for a basic development system
    - Install the Linux distribution of your choice
    - Minimal requirements to compile the Kernel
        - `sudo apt update`
        - `sudo apt-get install build-essential vim git cscope libncurses-dev libssl-dev bison flex`
    - Configuring email client for sending patches and responding to emails
        - Highly recommend using git send-email

Configuring Your Development System (2/3)

- Git Email Configuration
  - Configuring `git-email` to send patches is easy using the `sendemail` configuration option, once you have your smtp server configuration



```
[sendemail]
        smtpserver     = smtp.gmail.com
        smtpserverport = 587
        smtpencryption = tls
        smtpuser       = <username>
        smtppass       = <password>
```

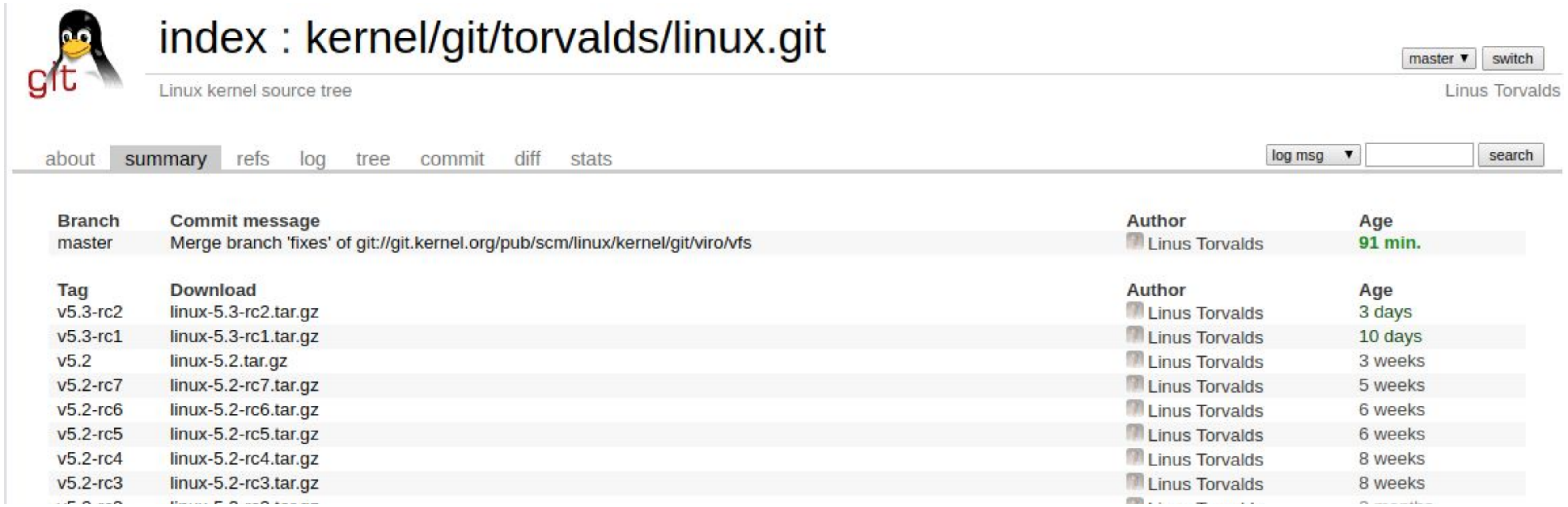
  - Place the above configuration in your `.gitconfig` file
  - Running `git send-email mypatch.patch` is all you have to do to send patches
    - `mypatch.patch` is generated by `git format-patch` command

Configuring Your Development System (3/3)

- Email Client Configuration
    - Refer to *Email clients info for Linux* for more details
    - Things to remember
        - Bottom post
        - Inline post
        - No HTML format
        - No signatures
        - No attachments
    - Refer to the Examples in *this document* for more information

Exploring Linux Kernel Sources

- The latest stable and mainline releases hosted on *The Linux Kernel Archives* web page
- Access the Linux mainline to explore the kernel sources

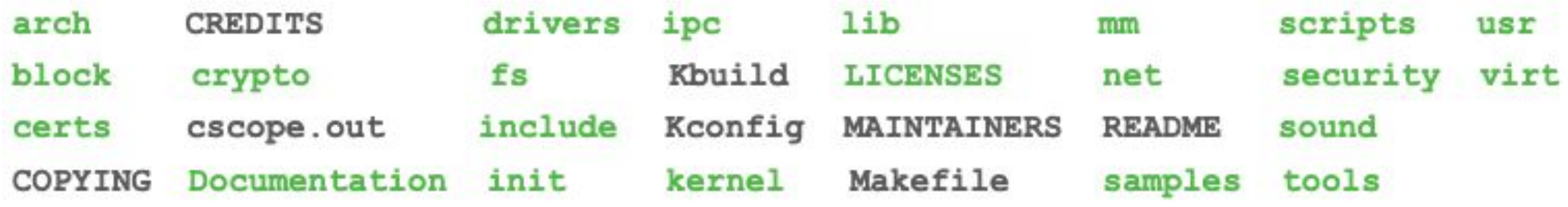


Branch	Commit message	Author	Age
master	Merge branch 'fixes' of git://git.kernel.org/pub/scm/linux/kernel/git/viro/vfs	Linus Torvalds	91 min.

Tag	Download	Author	Age
v5.3-rc2	linux-5.3-rc2.tar.gz	Linus Torvalds	3 days
v5.3-rc1	linux-5.3-rc1.tar.gz	Linus Torvalds	10 days
v5.2	linux-5.2.tar.gz	Linus Torvalds	3 weeks
v5.2-rc7	linux-5.2-rc7.tar.gz	Linus Torvalds	5 weeks
v5.2-rc6	linux-5.2-rc6.tar.gz	Linus Torvalds	6 weeks
v5.2-rc5	linux-5.2-rc5.tar.gz	Linus Torvalds	6 weeks
v5.2-rc4	linux-5.2-rc4.tar.gz	Linus Torvalds	8 weeks
v5.2-rc3	linux-5.2-rc3.tar.gz	Linus Torvalds	8 weeks

Cloning the Linux Mainline

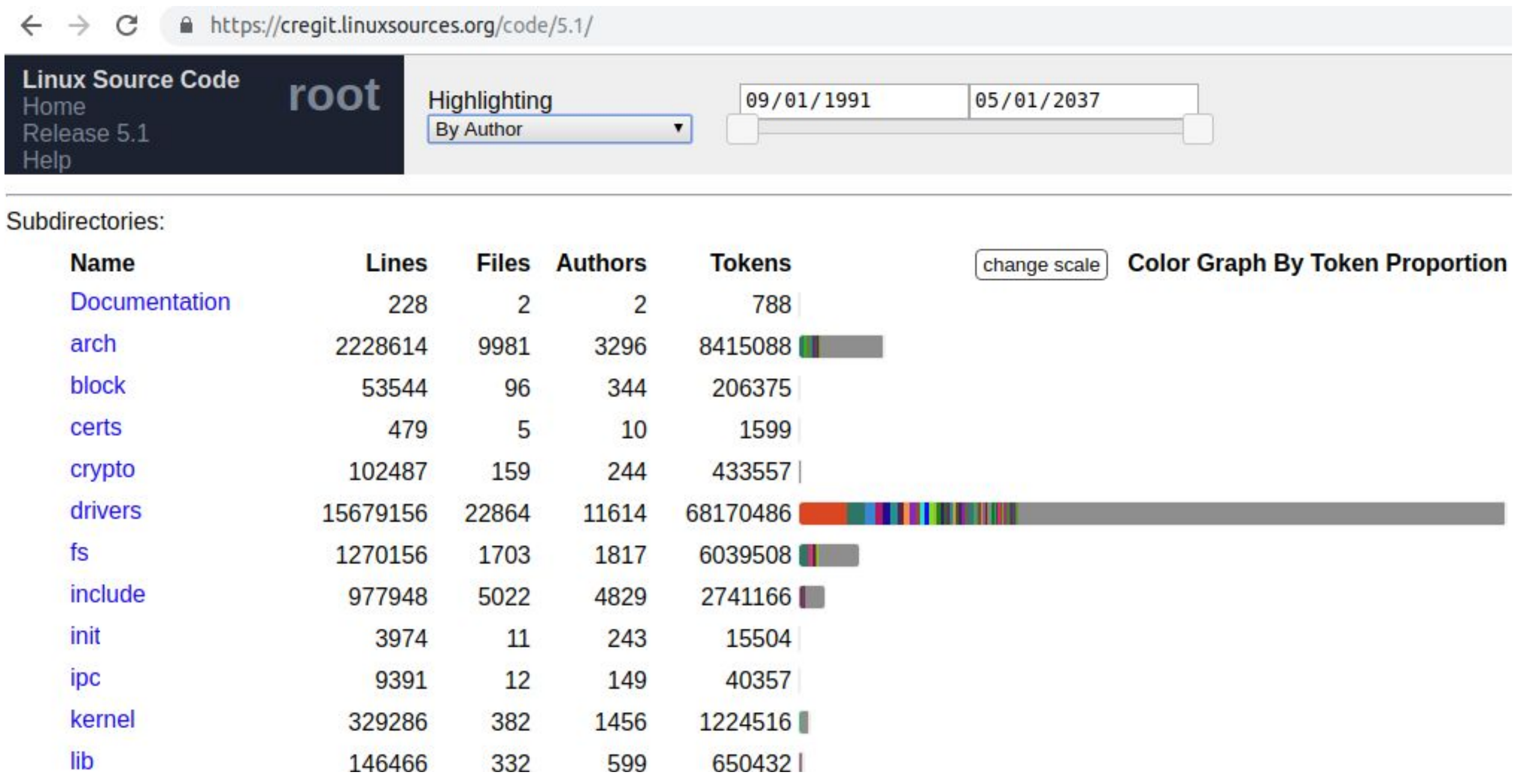
- Create a new directory named linux_mainline and populate it with the sources
  - `cd /linux_work`
  - `git clone \`
    `git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git linux_mainline`
  - `cd linux_mainline; ls -h`



```
arch     CREDITS        drivers  ipc      lib          mm       scripts   usr
block    crypto         fs       Kbuild   LICENSES     net      security  virt
certs    cscope.out     include  Kconfig  MAINTAINERS  README   sound
COPYING  Documentation  init     kernel   Makefile     samples  tools
```

What Is in the Root Directory?

- This screenshot presents a view of the Linux 5.1 release on cregit-Linux



Highlighting: By Author

Subdirectories:

Name	Lines	Files	Authors	Tokens
Documentation	228	2	2	788
arch	2228614	9981	3296	8415088
block	53544	96	344	206375
certs	479	5	10	1599
crypto	102487	159	244	433557
drivers	15679156	22864	11614	68170486
fs	1270156	1703	1817	6039508
include	977948	5022	4829	2741166
init	3974	11	243	15504
ipc	9391	12	149	40357
kernel	329286	382	1456	1224516
lib	146466	332	599	650432

Color Graph By Token Proportion

Exploring the Sources

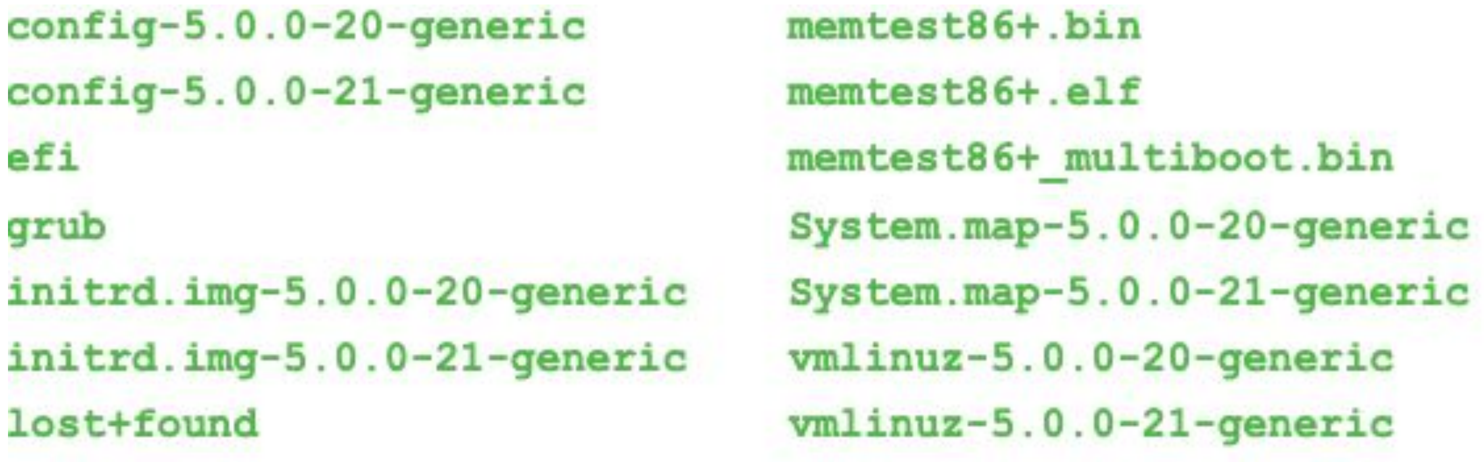
- You are encouraged to explore the sources. Take a look at below files, you will be using them in your everyday kernel development life.
    - Makefile and MAINTAINERS files in the main directory
    - scripts/get_maintainer.pl and scripts/checkpatch.pl
- Play with `cregit` or `git log` to look at the history of source files in each of the kernel areas
- Look at individual commits and generate a patch or two using
    - `git format-patch -1 <commit ID>`

Building and Installing Your First Kernel

- Cloning the Stable Kernel Git
  - `git clone \`
    `git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git linux_stable`
  - `cd linux_stable`
  - `git branch -a | grep linux-5`
    - `remotes/origin/linux-5.12.y`
    - `remotes/origin/linux-5.11.y`
    - `remotes/origin/linux-5.10.y`
  - `git checkout linux-5.12.y`

Building and Installing Your First Kernel

- Generate a kernel configuration file based on the current configuration
  - Using the distribution configuration file is the safest approach for the very first kernel install on any system
    - Copying from `/proc/config.gz` or `/boot`
  - `ls /boot`
  - `cp /boot/<config-5.0.0-21-generic> .config`



```
config-5.0.0-20-generic          memtest86+.bin
config-5.0.0-21-generic          memtest86+.elf
efi                              memtest86+_multiboot.bin
grub                             System.map-5.0.0-20-generic
initrd.img-5.0.0-20-generic      System.map-5.0.0-21-generic
initrd.img-5.0.0-21-generic      vmlinuz-5.0.0-20-generic
lost+found                       vmlinuz-5.0.0-21-generic
```

Building and Installing Your First Kernel

- Generate a kernel configuration file based on the current configuration
  - `make olddefconfig`
- Another way: Creates a configuration file based on the list of modules currently loaded on your system
  - `lsmod > /tmp/my-lsmod`
  - `make LSMOD=/tmp/my-lsmod localmodconfig`
- It is time to compile the kernel
  - `make -j3 all`
  - `j` option specifies the number of jobs (make commands) to run simultaneously
- Compiling a single source file: `make path/file.o`
- Compiling at the directory level: `make path`

Linux Kernel Configuration

- Linux kernel is completely configurable
- Drivers can be:
    - Disabled
    - Built into the kernel (vmlinux image) to be loaded at boot time
    - Built as a module to be loaded as needed using `modprobe`
- Should configure drivers as modules, to avoid large kernel images
    - Modules (.ko files) can be loaded when the kernel detects hardware that matches the driver
- `modprobe`
    - Add/remove a module from the Linux kernel
- `sysctl`
    - List/modify kernel parameters at runtime (listed under `/proc/sys/`)

Writing Your First Kernel Patch

- Creating a new Branch in the linux_mainline repository
  - `git checkout -b first-patch`
- Update the kernel
  - `git fetch origin`
- Making changes to a driver
  - Run `lsmod` to see the modules loaded on your system, and pick a driver to change
  - One driver that's included in all VM images is the `e1000` driver, the Intel ethernet driver
  - Run `git grep` to look for `e1000` files
    - `git grep e1000 -- '*Makefile'`

Writing Your First Kernel Patch

- Make a small change to the probe function of the e1000 driver
  - `nano drivers/net/ethernet/intel/e1000/e1000_main.c`

```
static int e1000_probe(struct pci_dev *pdev, const struct
pci_device_id *ent) {
        struct net_device *netdev;
        struct e1000_adapter *adapter;
        struct e1000_hw *hw;
        printk(KERN_DEBUG "I can modify the Linux kernel!\n");
        static int cards_found = 0;
```

- Compile, install, test your changes
  - `make -j3`
  - `sudo make modules_install install`
  - `dmesg | less`

Sending a Patch for Review

- Committing changes, and view your commit
  - `git commit -s -v`
  - `git show HEAD`
  - `git log --pretty=oneline --abbrev-commit`
- The `get_maintainer.pl` script tells you whom to send the patch to
  - `git show HEAD | scripts/get_maintainer.pl`
- Creating a patch
  - `git format-patch -1 <commit ID> --to=<maintainer email> --cc=<maintainer email>`
- Sending patches
  - `git send-email <patch_file>`

Applying Patches

- Linux kernel patch files are text files that contain the differences from the original source to the new source
  - `git apply --index file.patch`

The Review Process

- Your patch will get comments from reviewers with suggestions for improvements
    - In some cases, learning to know more about the change itself
- Please be patient and wait for a minimum of one week before requesting a response.
    - During merge windows and other busy times, it might take longer than a week to get a response
    - Make sure you sent the patch to the right recipients
- Best Practices for Sending Patches

Testing Open Source Software

- Testing is very important when it comes to any software, not just the Linux kernel
- Ensuring software is stable without regressions before the release helps avoid debugging and fixing customer/user-found bugs after the release
    - It costs more in time and effort to debug and fix a customer-found problem
- In the open source development model, developers and users share the testing responsibility
- Users and developers that are not familiar with the new code may be more effective at testing a new piece of code than the original author of that code

Automated Build Bots and Continuous Integration Test Rings

- Maintainers and developers can request their repositories to be added to the linux-next integration tree and the 0-day build bot
- 0-day build bot can pull in patches and run build tests on several configurations and architectures
    - Helpful in finding compile errors and warnings that might show up on other architectures that developers might not have access to
- Continuous Integration (CI) rings are test farms that host several platforms and run boot tests and Kernel Self-tests on stable, linux-next, and mainline trees
    - Kernel CI Dashboard
    - 0-Day - Boot and Performance issues
    - 0-Day - Build issues
    - Linaro QA
    - Buildbot

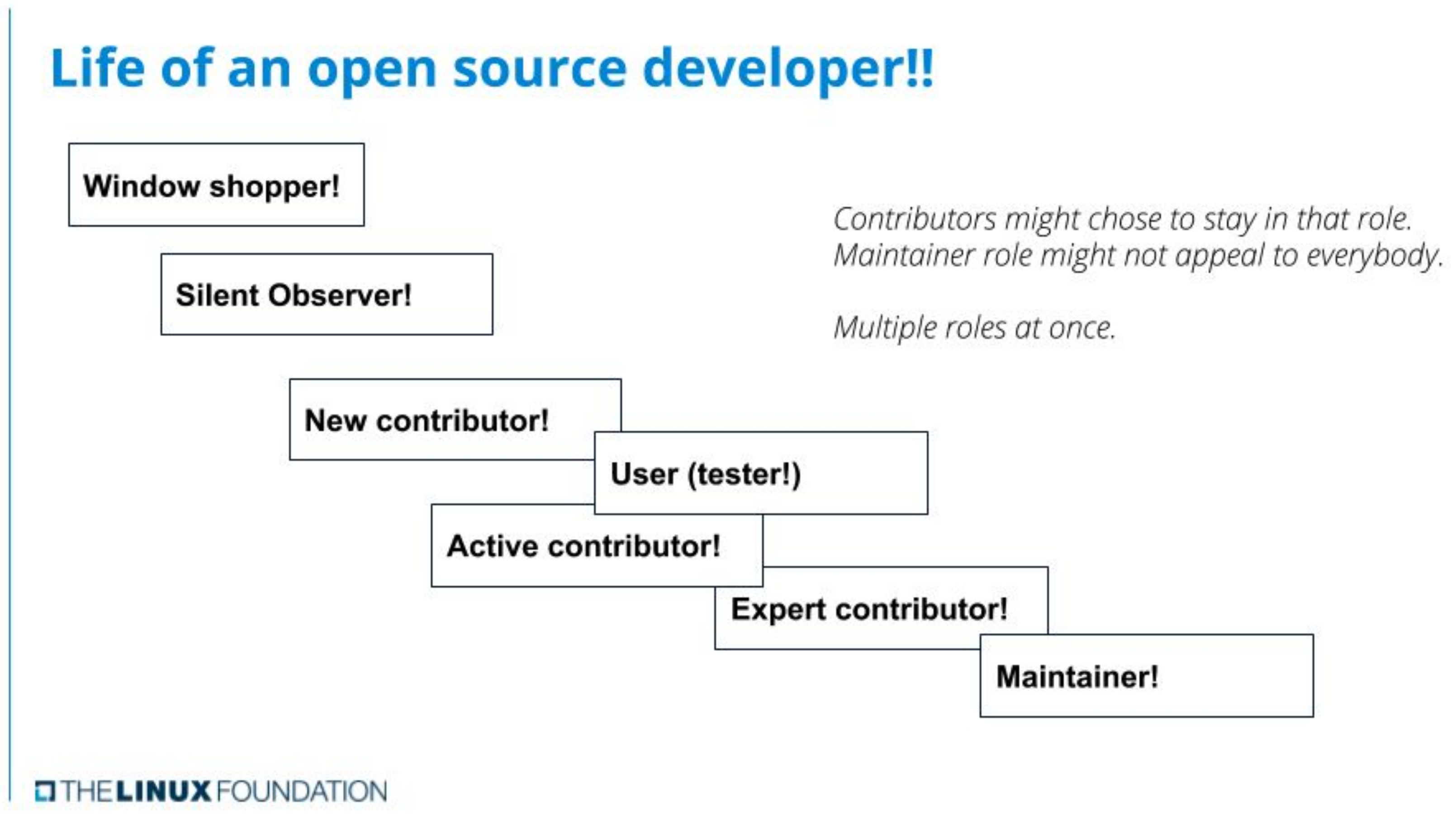
Kernel Debugging (1/2)

- Debugging is an art, and not a science. There is no step-by-step procedure or a single recipe for success when debugging a problem
- Asking the following questions can help to understand and identify the nature of the problem and how best to solve it:
    - Is the problem easily reproducible?
    - Is there a reproducer or test that can trigger the bug consistently?
    - Are there any panic, or error, or debug messages in the dmesg when the bug is triggered?
    - Is reproducing the problem time-sensitive?

Kernel Debugging (2/2)

- What's in a Panic Message?
    - Debugging Analysis of Kernel panics and Kernel oopses using System Map
    - Understanding a Kernel Oops!
- Decode and Analyze the Panic Message
    - decode_stacktrace: make stack dump output useful again
- Use Event Tracing to Debug
    - Event Tracing page in the Linux Kernel Documentation
- Additional resources
    - Hunt bugs
    - Bisecting a bug
    - Dynamic debugging
    - Contributors to the Linux Kernel.
    - Who Made That Change and When: Using Cregit for Debugging

Life of an Open Source Developer



Life of an open source developer!!

Contributors might chose to stay in that role.
Maintainer role might not appeal to everybody.

Multiple roles at once.

How does one become a maintainer?

- There are many paths to choose from. A few paths that lead you there
  - Write a new driver
  - Adopt an orphaned driver or a module (Hint: Check the MAINTAINERS file).
  - Clean up a staging area driver to add it to the mainline
  - Work towards becoming an expert in an area of your interest by fixing bugs, doing reviews, and making contributions.
  - Find a new feature or an enhancement to an existing feature that is important to the kernel and its ecosystem
- Remember that new people bringing new and diverse ideas make the kernel better. We, as a community, welcome and embrace new ideas!

Continue Your Kernel Journey

- Participate in the Stable Release Process
  - Subscribe to the Stable release mailing list
  - Stable release maintainers send stable patches for testing with information on where to download the patch from
  - Download the patches or clone repository specified in the email
  - Compile and apply the patches
- Enhance and Improve Kernel Documentation

Continue Your Kernel Journey

- Contribute to the Kernel - Getting Started
    - Subscribe to the Linux Kernel mailing list for the area of your interest
    - Follow the development activity reading the Linux Kernel Mailing List Archives
    - Join the #kernelnewbies IRC channel on the OFTC IRC network
    - Join the #linux-kselftest, #linuxtv, #kernelci, or #v4l IRC channels on freenode
    - Find spelling errors in kernel messages
    - Static code analysis error fixing
    - Bug reports and fixing
    - Etc.
- Linux Foundation Events Calendar
- LF Live: Mentorship Series