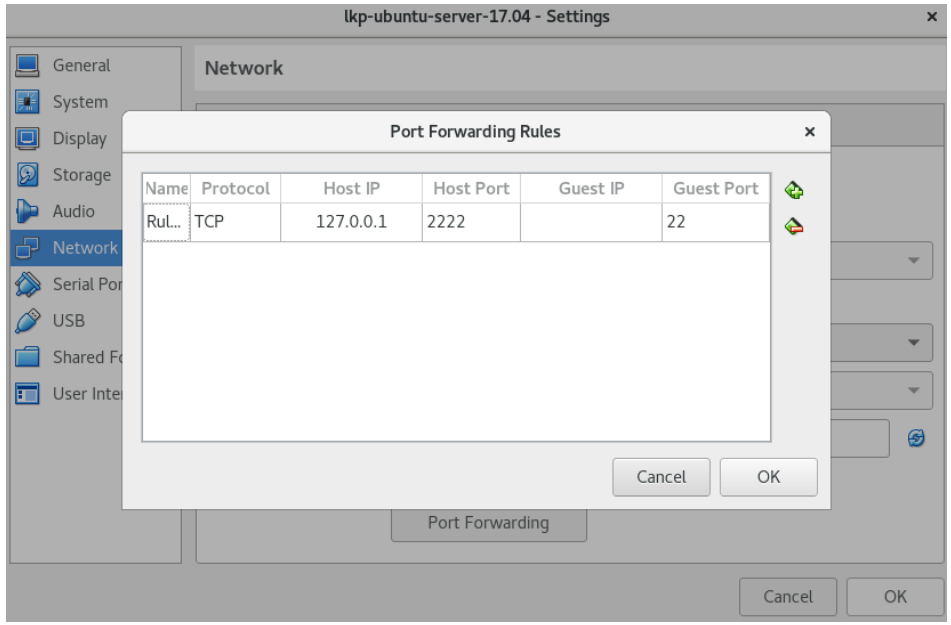
Building and exploring Linux kernel

Dongyoon Lee

Did you succeed in installing Linux on VirtualBox?



- Add a port forwarding rule on VirtualBox

Why software tools are important?

- Linux source code is huge and evolves very fast
  - 27 million lines of code (LoC) ← 1,600 developers / release

```
✔ ~/workspace/research/linux [v5.15]
$ tree .
├── arch
│   ├── alpha
│   │   ├── boot
│   │   │   ├── bootloader.lds
│   │   │   ├── bootp.c
    ...
├── lib
│   ├── irqbypass.c
│   ├── Kconfig
│   └── Makefile
└── Makefile

7468 directories, 75689 files
```

Today's lecture

- Tools
  - Version control : `git` , `tig`
  - Configure, build, and install the kernel : `make`
  - Explore the code : `cscope` , `ctags`
  - Editor : `vim` , `emacs`
  - Screen : `tmux`
- Kernel vs. user programming

Obtaining the kernel source code

- Tar ball
  - https://www.kernel.org/
- Linus's git repository
  - git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
- Github mirror of Linus's git repository
  - https://github.com/torvalds/linux.git
- Let's explore above web sites!

Version control: `git`

- **Git** is a version control software
  - tracking changes in computer files
- Initially developed by Linus Torvalds for development of the Linux kernel
  - Extensively using in many other software development
  - Github https://github.com/ is a `git` service provider
- Distributed revision control system
  - Every git directory on every computer is a full-fledged repository with complete history

Essential `git` commands

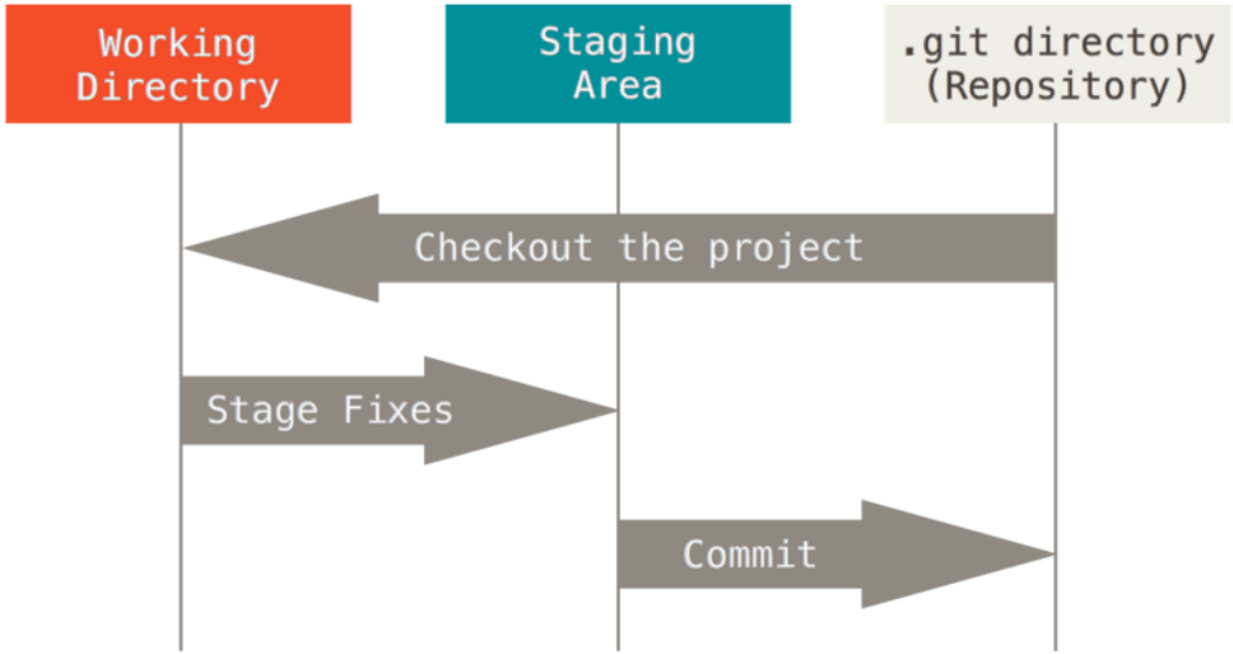
```
$ # 1. install and configure
$ sudo dnf install git # sudo apt-get install git
$ git config --global user.name "John Doe"  # set your name and email for history
$ git config --global user.email johndoe@example.com

$ # 2. creat a repository
$ git init                                          # create a new local repo
$ git clone https://github.com/torvalds/linux.git # clone an existing repo

$ # 3. tags
$ git tag            # list all existing tags
$ git checkout v5.15 # checkout the tagged version

$ # 4. commit history (or use tig for prettier output)
$ git log            # show all commit history
$ git log <file>     # show changes over time for a file
$ git blame <file>   # who changed what and when in <file>
```

Essential `git` commands

```
$ # 5. local changes
$ git status          # show changed files
$ git diff            # show changed lines
$ git add <file>      # add <file> to the next commit
$ git commit          # commit previously staged files to my local repo

$ # 6. publish and update
$ git push            # publish a committed local changes to a remote repo
$ git pull            # update a local repo
```

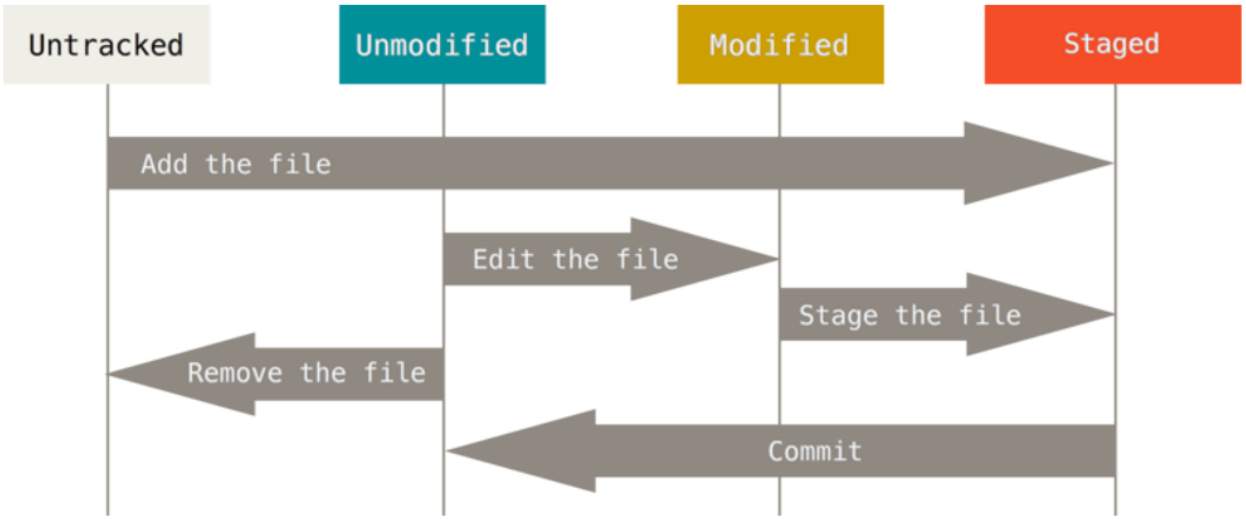
- Many useful git tutorials:
  - Atlassian, Github, TutorialsPoint, Linux kernel, Pro Git

`git` workflow



- Source: [Pro Git]

`git` workflow



- Source: Pro Git

The kernel source tree

```
$ git clone https://github.com/torvalds/linux.git # clone the kernel repo
$ cd linux; git checkout v5.15  # checkout v5.15
$
$ tree -d -L 2      # list top two-level directories
├── arch            # * architecture dependent code
│   ├── arm         #   - ARM architecture
│   └── x86         #   - Intel/AMD x86 architecture
├── block           # * block layer: e.g., IO scheduler
├── Documentation   # * design documents
├── drivers         # * device drivers
│   └── nvme        #   - NVMe SSD
├── fs              # * virtual file system (VFS)
│   ├── ext4        #   - ext4 file system
│   └── xfs         #   - XFS file system
├── include         # * include files
│   ├── linux       #   - include files for kernel
│   └── uapi        #   - include files for user-space tools
├── init            # * bootig: start_kernel() at main.c
├── ipc             # * IPC: e.g., semaphore
│   ...
```

The kernel source tree

```
│   ...
├── kernel          # * core features of the kernel
│   ├── locking     #   - locking: e.g., semaphore, mutex, spinlock
│   └── sched       #   - task scheduler
├── lib             # * common library: e.g., red-black tree
├── mm              # * memory management: e.g., memory allocation, paging
├── net             # * network stack
│   ├── ipv4        #   - TCP/IPv4
│   └── ipv6        #   - TCP/IPv6
├── security        # * security framework
│   └── selinux     #   - selinux
├── tools           # * user-space tools
│   └── perf        #   - perf: performance profiling tool
└── virt            # * virtualization
    └── kvm         #   - KVM type-2 hypervisor

615 directories
```

Build the kernel

1. Configuring the kernel
    - Configuration file defining compilation options (~ 3700 for x86)
2. Compiling the kernel
    - Compile and link the kernel source code
3. Installing the new kernel
    - Install compiled new kernel image to a system

- `make help` to see other make options
- Ref: Documentation/admin-guide/README.rst

Configure the kernel

- `make menuconfig`
  - Need *libncurses*
    - sudo dnf install ncurses-devel # Fedora/CentOS/RedHat
    - sudo apt-get install libncurses5-dev # Debian/Ubuntu

Configure the kernel

- `make defconfig`
  - Generate the default configuration of running platform
  - linux/arch/x86/configs/x86_64_defconfig
- `make oldconfig`
  - Use the configuration file of running kernel
  - Will ask about new configurations
    - If you are not sure, choose default options
- `make localmodconfig`
  - Update current config disabling modules not loaded

Kernel configuration file: `.config`

- `.config` file is at the root of the kernel source
  - preprocessor flags in the source code

```
# linux/.config
# CONFIG_XEN_PV is not set
CONFIG_KVM_GUEST=y
CONFIG_XFS_FS=m
```

```
/* linux/arch/x86/kernel/cpu/hypervisor.c */
static const __initconst struct hypervisor_x86 * const hypervisors[] =
{
#ifdef CONFIG_XEN_PV
    &x86_hyper_xen_pv,
#endif
#ifdef CONFIG_KVM_GUEST
    &x86_hyper_kvm,
#endif
};
```

Compile the kernel

1. Compile the kernel: `make`
   - Compile the kernel source code
   - Compiled kernel image: linux/arch.x86/boot/bzImage
2. Compile modules: `make modules`

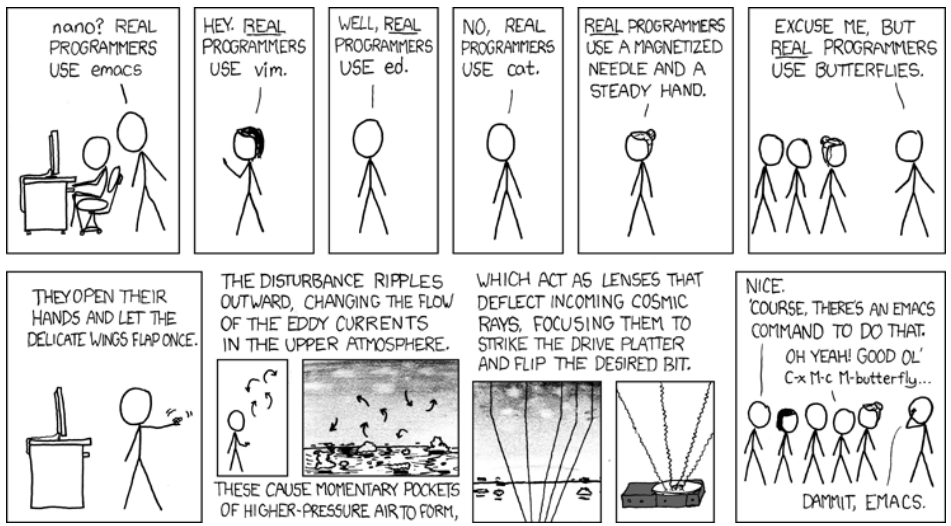
- Parallel `make`
  - `make <target> -j<number of CPUs to use>`
  - E.g., `make -j4`

Install the new kernel

```
# Install the new kernel modules (if you change modules)
$ sudo make modules_install
$ ls /lib/modules/

# Install the new kernel image
$ sudo make install
$ ls /boot/*5.15*
/boot/config-5.15.0      /boot/initrd.img-5.15.0
/boot/System.map-5.15.0  /boot/vmlinuz-5.15.0

# Reboot the machine
$ sudo reboot

# Check if a system boots with the new kernel
$ uname -a
Linux dongyoon 5.15.0 #1 SMP Mon Jan 25 22:56:41 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux

# See kernel log
$ dmesg
$ dmesg -w # wait for new kernel messages
```

Editor

- There are many good editors
  - `vim`, `emacs`



- Source: https://xkcd.com/378/

Exploring the code

- Linux Cross Reference (LXR)
- `cscope`
- `vim` with `cscope` or `ctags`
- `emacs` with `cscope`
- …

Linux Cross Reference (LXR)

- Code indexing tool with a web interface
  - Don't install it! One instance is running here:
  - http://lxr.free-electrons.com/
- Allows to:
  - Browse the code of different Linux versions
  - Search for identifiers (functions, variables, etc.)
  - Quickly lookup a function declaration/definition

Linux Cross Reference (LXR)

Linux Cross Reference (LXR)

Linux Cross Reference (LXR)

```
533                                     static_command_line, __start___param,
534                                     __stop___param - __start___param,
535                                     -1, -1, &unknown_bootoption);
536         if (!IS_ERR_OR_NULL(after_dashes))
537                 parse_args("Setting init args", after_dashes, NULL, 0, -1, -1,
538                            set_init_arg);
539
540         jump_label_init();
541
542         /*
543          * These use large bootmem allocations and must precede
544          * kmem_cache_init()
545          */
546         setup_log_buf(0);
547         pidhash_init();
548         vfs_caches_init_early();
549         sort_main_extable();
550         trap_init();
551         mm_init();
552
553         /*
554          * Set up the scheduler prior starting any interrupts (such as the
555          * timer interrupt). Full topology setup happens at smp_init()
556          * time - but meanwhile we still have a functioning scheduler.
557          */
558         sched_init();
559         /*
560          * Disable preemption - early bootup scheduling is extremely
561          * fragile until we cpu_idle() for the first time.
562          */
563         preempt_disable();
564         if (WARN(!irqs_disabled(),
565                  "Interrupts were enabled *very* early, fixing it\n"))
566                 local_irq_disable();
567         idr_init_cache();
568         rcu_init();
569
570         /* trace_printk() and trace points may be used after this */
571         trace_init();
572
573         context_tracking_init();
574         radix_tree_init();
575         /* init some links before init_ISA_irqs() */
```

Click on a function call to search for the function

Linux Cross Reference (LXR)

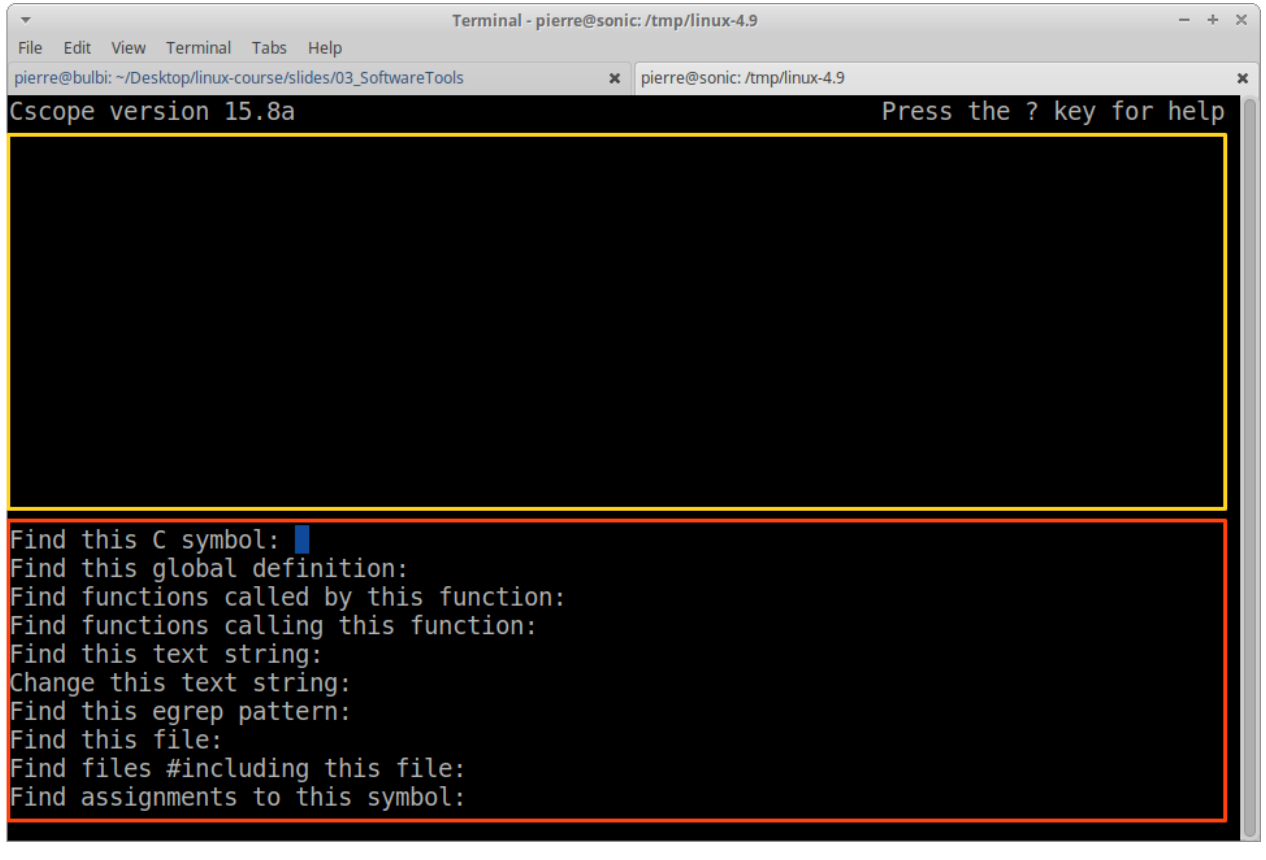
`cscope`

- Command line tool to browse (potentially large) C codebases
- Installation: `sudo {apt-get|dnf} install cscope`
- Build cscope database
  - `cd linux; KBUILD_ABS_SRCTREE=1 ARCH=x86 make cscope` # only for x86
  - Need to rebuild after code changes
- Although `cscope -R` is the common way to build cscope database, `make cscope` is optimized for the kernel source code

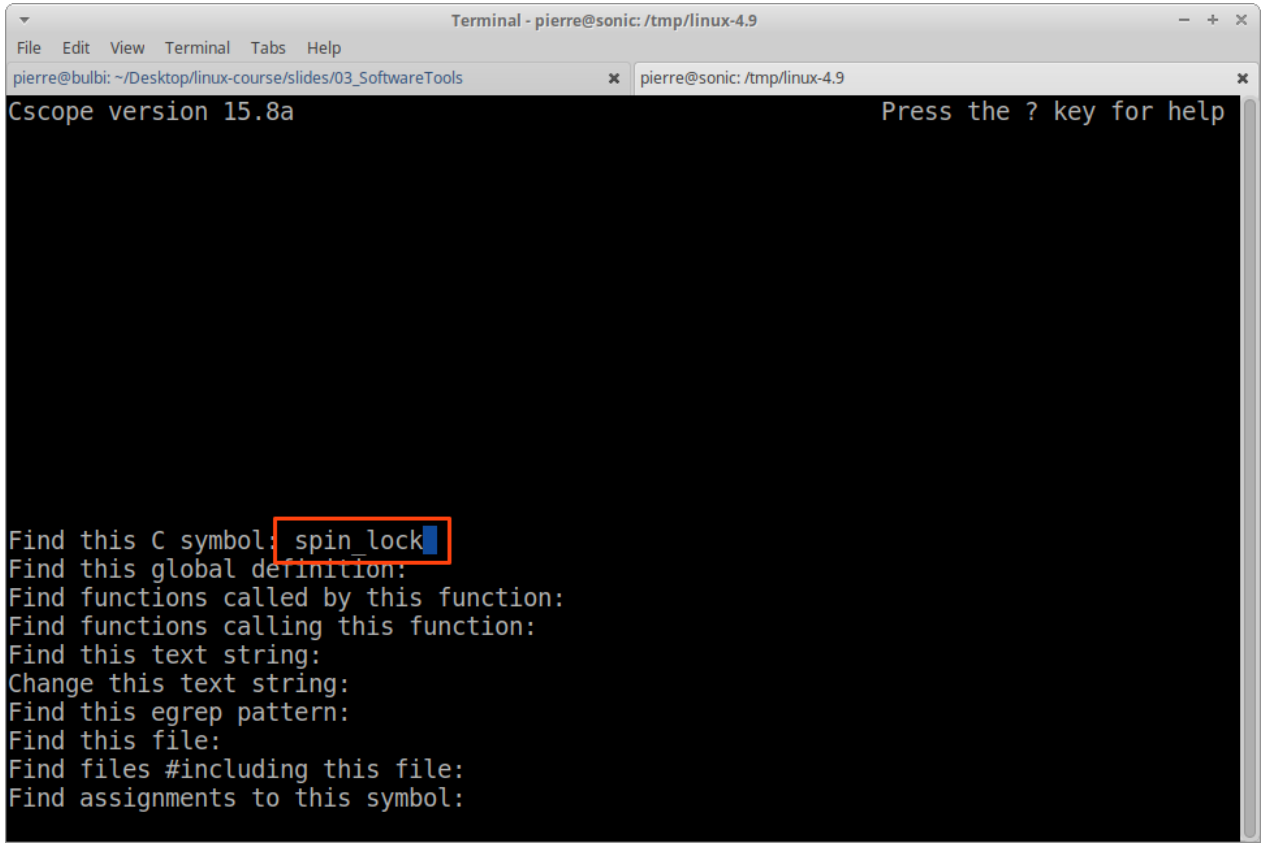
`cscope`

- Search for:
  - C identifier occurrences (variable name, function name, typedef/struct, label)
  - Functions/variables definitions
  - Functions called by/calling function *f*
  - Text string
- Terminating cscope: Ctrl-d

cscope


Terminal - pierre@sonic: /tmp/linux-4.9
File Edit View Terminal Tabs Help
pierre@bulbi: ~/Desktop/linux-course/slides/03_SoftwareTools
pierre@sonic: /tmp/linux-4.9
Cscope version 15.8a
Press the ? key for help
Find this C symbol:
Find this global definition:
Find functions called by this function:
Find functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
Find assignments to this symbol:

cscope


Terminal - pierre@sonic: /tmp/linux-4.9
File Edit View Terminal Tabs Help
pierre@bulbi: ~/Desktop/linux-course/slides/03_SoftwareTools
pierre@sonic: /tmp/linux-4.9
Cscope version 15.8a
Press the ? key for help
Find this C symbol: spin_lock
Find this global definition:
Find functions called by this function:
Find functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
Find assignments to this symbol:

cscope

Terminal - pierre@sonic: /tmp/linux-4.9

File Edit View Terminal Tabs Help

pierre@bulbi: ~/Desktop/linux-course/slides/03_SoftwareTools | pierre@sonic: /tmp/linux-4.9

```
C symbol: spin_lock

  File                    Function                  Line
0 platsmp.c               <global>                   287 spin_lock(&boot_lock);
1 bus.c                   <global>                  1049 spin_lock(&device_klis
                                                         t->k_lock);
2 platform.c              <global>                   697 spin_lock(&drv->driver
                                                         .bus->p->klist_drivers
                                                         .k_lock);
3 runtime.c               <global>                   279 spin_lock(&dev->power.
                                                         lock);
4 omap_gem.c              <global>                  1256 spin_lock(&sync_lock);

* Lines 1-6 of 9743, 9738 more - press the space bar to display more *

Find this C symbol:
Find this global definition:
Find functions called by this function:
Find functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
Find assignments to this symbol:
```

cscope

Terminal - pierre@sonic: /tmp/linux-4.9

File Edit View Terminal Tabs Help

pierre@bulbi: ~/Desktop/linux-course/slides/03_SoftwareTools | pierre@sonic: /tmp/linux-4.9

```
Cscope version 15.8a                                    Press the ? key for help



Find this C symbol:
Find this global definition: spin_lock
Find functions called by this function:
Find functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
Find assignments to this symbol:
```

cscope

Terminal - pierre@sonic: /tmp/linux-4.9

File Edit View Terminal Tabs Help

pierre@bulbi: ~/Desktop/linux-course/slides/03_SoftwareTools | pierre@sonic: /tmp/linux-4.9

```
Global definition: spin_lock

  File          Line
0 aic79xx_osm.h 352 spinlock_t spin_lock;
1 aic7xxx_osm.h 356 spinlock_t spin_lock;
2 comedidev.h   177 spinlock_t spin_lock;
3 spinlock.h    300 static   always inline void spin lock(spinlock_t *lock)

Find this C symbol:
Find this global definition:
Find functions called by this function:
Find functions calling this function:
Find this text string:
Change this text string:
Find this egrep pattern:
Find this file:
Find files #including this file:
Find assignments to this symbol:
```

cscope

Terminal - pierre@sonic: /tmp/linux-4.9

File Edit View Terminal Tabs Help

pierre@bulbi: ~/Desktop/linux-course/slides/03_SoftwareTools | pierre@sonic: /tmp/linux-4.9

```
static __always_inline raw_spinlock_t *spinlock_check(spinlock_t *lock)
{
        return &lock->rlock;
}

#define spin_lock_init(_lock)                           \
do {                                                    \
        spinlock_check(_lock);                          \
        raw_spin_lock_init(&(_lock)->rlock);            \
} while (0)

static __always_inline void spin_lock(spinlock_t *lock)
{
        raw_spin_lock(&lock->rlock);
}

static __always_inline void spin_lock_bh(spinlock_t *lock)
{
        raw_spin_lock_bh(&lock->rlock);
}

static __always_inline int spin_trylock(spinlock_t *lock)
{
        return raw_spin_trylock(&lock->rlock);
"include/linux/spinlock.h" [readonly] 419L, 11597C                300,1         72%
```

`vim` with `cscope` or `ctags`

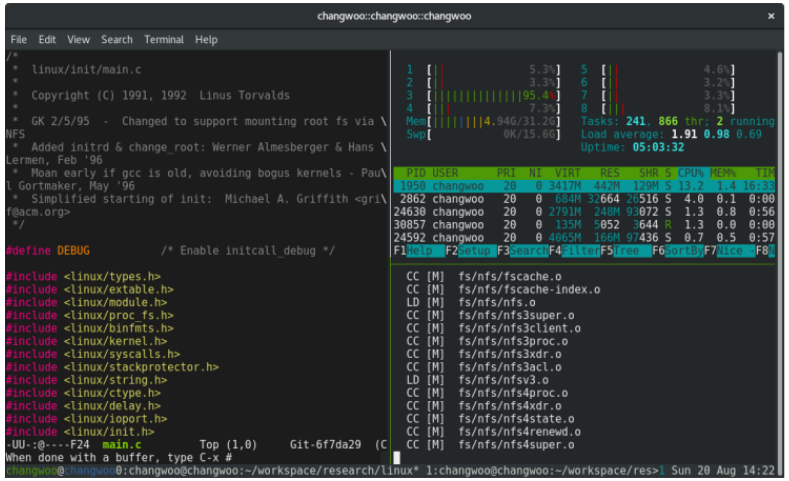
- `vim` can use the tag database of `cscope`, as well as `ctags`
  - `sudo apt-get install cscope exuberant-ctags`
  - `sudo dnf install cscope ctags`
- Generate the database
  - `cd linux; make cscope tags -j2`
- Launch vim
  - `vim init/main.c`

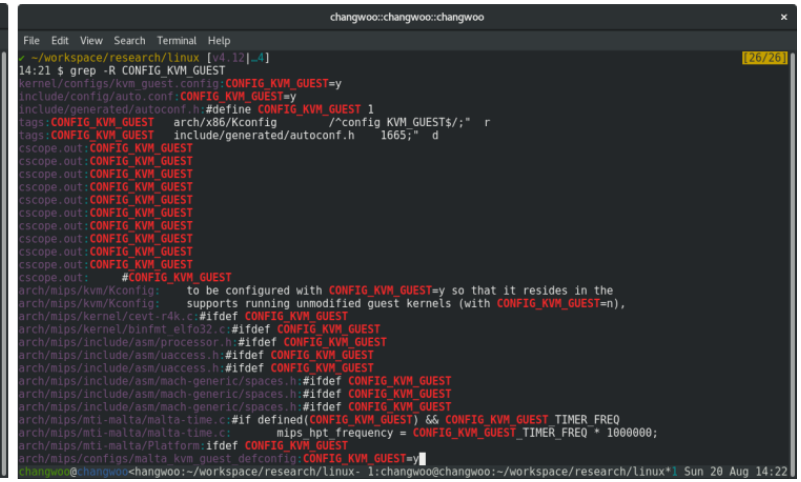
`vim` with `cscope` or `ctags`

- Search for function definition/variable declaration:
  - `:tag start_kernel` or `:cs find global start_kernel`
- Help for `ctags` and `cscope`
  - `:help tag` or `:help cs`
- Another way to find a function definition/variable declaration:
  - Put the cursor on the symbol name and press `Ctrl+]`
- To navigate back and forth between file:
  - `:bp` or `:bn`

Screen: `tmux`

- `tmux` is a tool to manage virtual consoles





- Ref: [A tmux Primer]

Essential `tmux` commands

- `tmux` : start a new tmux session
- `Ctrl-b %` : split a pane vertically
- `Ctrl-b "` : split a pane horizontally
- `Ctrl-b o` : move to the next pane
- `Ctrl-b z` : zoom (or unzoom) a pane
- `Ctrl-b c` : create a new window
- `Ctrl-b N` : go to window N (0~9)
- `Ctrl-b d` : detach from a session
- `tmux a` : attach to an existing session

Kernel vs. user programming

- No libc or standard headers
  - Instead the kernel implements lots of libc-like functions
- Examples
  - `#include <string.h>` → `#include <linux/string.h>`
  - `printf("Hello!")` → `printk(KERN_INFO "Hello!")`
  - `malloc(64)` → `kmalloc(64, GFP_KERNEL)`

Kernel vs. user programming

- Use GCC extensions
- Inline functions
  - `static inline void func()`
- Inline assembly: less than 2%
  - `asm volatile("rdtsc" : "=a" (l), "=d" (h));`
- Branch annotation: hint for better optimization
  - `if (unlikely(error)) {...}`
  - `if (likely(success)) {...}`

Kernel vs. user programming

- No (easy) use of floating point
- Small, fixed-size stack: 8 KB (2 pages) in x86
- No memory protection
  - `SIGSEGV` → **kernel panic (oops)**
- An example of kernel oops

Kernel vs. user programming

- Synchronization and concurrency
  - Muti-core processor → synchronization among tasks
    - A kernel code can execute on two more processors
  - Preemptive multitasking → synchronization among tasks
    - A task can be scheduled and re-scheduled at any time
  - Interrupt → synchronization with interrupt handlers
    - Can occur in the midst of execution (e.g., accessing resource)
    - Need to synchronize with interrupt handler

Linux kernel coding style

- Indentation: 1 tab → 8-character width (not 8 spaces)
- No CamelCase use underscores: `SpinLock` → `spin_lock`
- Use C-style comments: /* use this style */ // not this
- Line length: 80 column
- **Write code in a similar style with other kernel code**
- Ref: Documentation/process/coding-style.rst

Linux kernel coding style

```
/*
 * a multi-lines comment
 * (no C++ '//' !)
 */

struct foo {
        int member1;
        double member2;
}; /* no typedef ! */

#ifdef CONFIG_COOL_OPTION
int cool_function(void) {
    return 42;
}
#else
int cool_function(void) { }
#endif /* CONFIG_COOL_OPTION */
```

Linux kernel coding style

```
void my_function(int the_param, char *string, int a_long_parameter,
    int another_long_parameter)
{
    int x = the_param % 42;

    if (!the_param)
        do_stuff();

    switch (x % 3) {
    case 0:
        do_some_stuff();
        cool_function();
        break;
    case 1:
        /* Fall through */
    default:
        do_other_stuff();
        cool_function();
    }
}
```

Summary of tools

- Version control : `git` , `tig`
- Configure the kernel : `make oldconfig`
- Build the kernel : `make -j8; make modules -j8`
- Install the kernel : `make install; make modules_install`
- Explore the code : `make cscope tags -j2` ; `cscope` , `ctags`
- Editor : `vim` , `emacs`
- Screen : `tmux`

Other useful sources II

- Documentation directory: the most up-to-date design documents
- The Linux Kernel Documentation: the extensive documents extracted from kernel source
- Linux Weekly News: easy explanation of recently added kernel features
- Linux Inside: textbook-style description on kernel subsystems
- Kernel newbies: useful information for new kernel developers
- Linux Kernel API Manual
- Kernel Recipes
- kernel planet

Next actions

- Master the essential tools, seriously
  - editor: `vim`, `emacs`
  - code navigation: `cscope`, `ctags`
  - version control: `git`, `tig`
  - terminal: `ssh`, `tmux`
- Useful lecture videos: Vim, tmux,ssh, Git

Next lecture

- Isolation and system call
- Explore how following three system calls are implemented in the kernel

```
fd = open("out", 1);
write(fd, "hello\n", 6);
pid = fork();
```