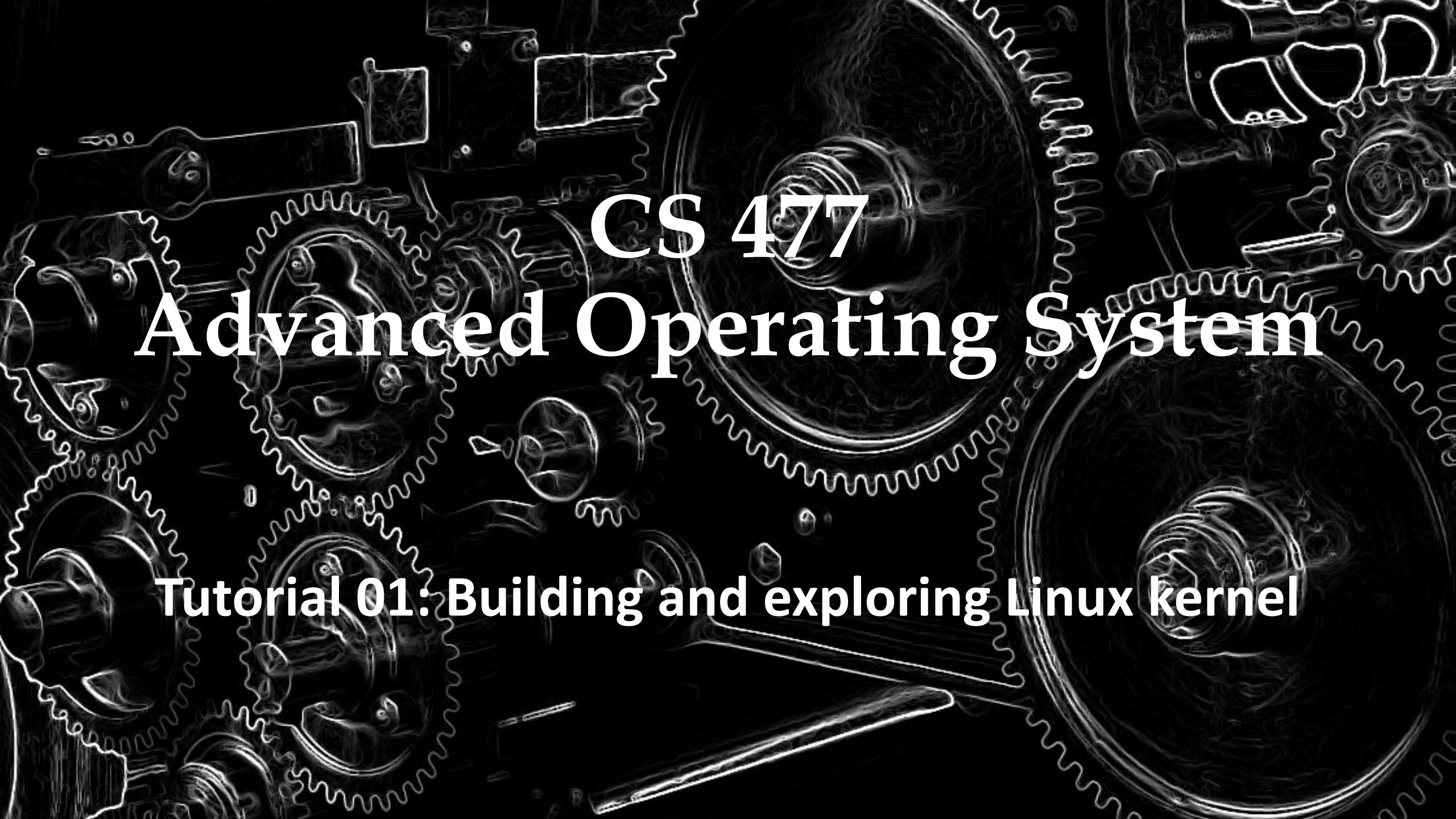


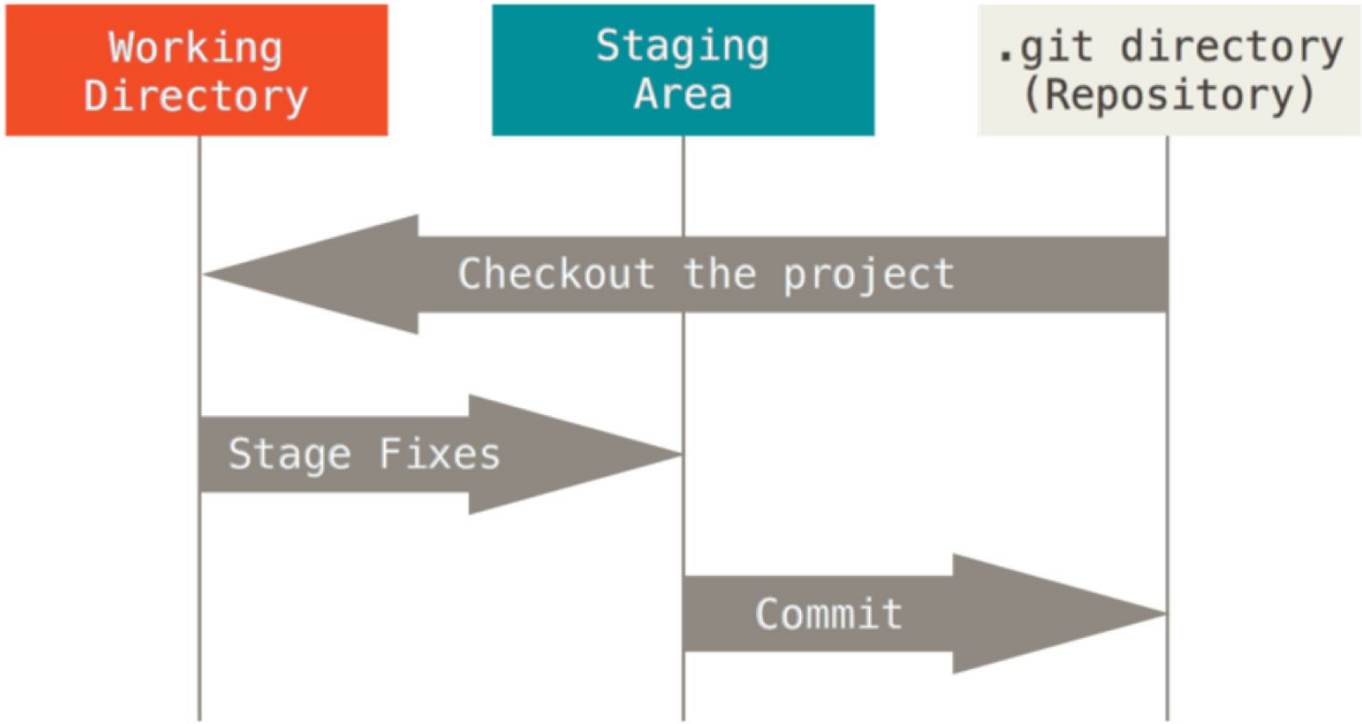
CS 477
Advanced Operating System

Tutorial 01: Building and exploring Linux kernel

Today's tutorial

- Tools
  - **Version control**: `git`
  - **Configure, build, and install the kernel**: `make`
  - **Screen**: `tmux`
- Kernel vs. user space programming

Git workflow



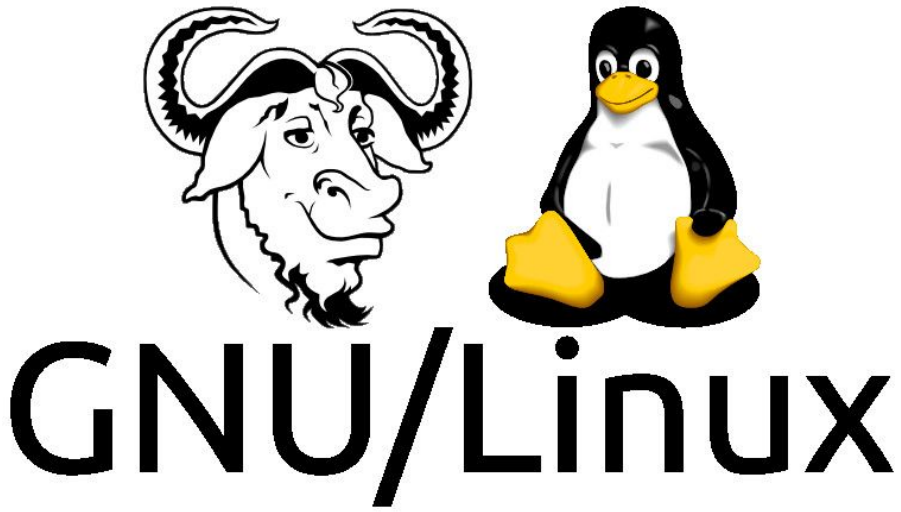
- Source: Pro Git

Ubuntu or Linux?

- Who uses any kind-of Linux ?
- What is Ubuntu & Linux?
  - Ubuntu is a **distribution** not a kernel.
  - (Sometimes) different package managers
  - (Sometimes) different user interfaces
  - Everyone can create their own distribution
- They have a common base: the **Linux** kernel

Basic Linux concepts ?

- Are these tools part of a Linux kernel? What are these tools ?
  - ~ ls
  - ~ mkdir
  - ~ touch
- These are not a part of the Linux kernel. These are **GNU** core utilities.And this also why some people refer to the entire system as "GNU/Linux".



Obtaining the Linux kernel source code

- Tar ball
  - https://www.kernel.org
- Linus's git repository
  - 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 see the above websites …

Version control: git

- **Git** is a version control software
  - Tracking changes in project files
  - Easier collaboration
- 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

Essential git commands

```
$ # 1. Install and configure
$ sudo apt 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. Create or clone 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 v6.15 # 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>
```

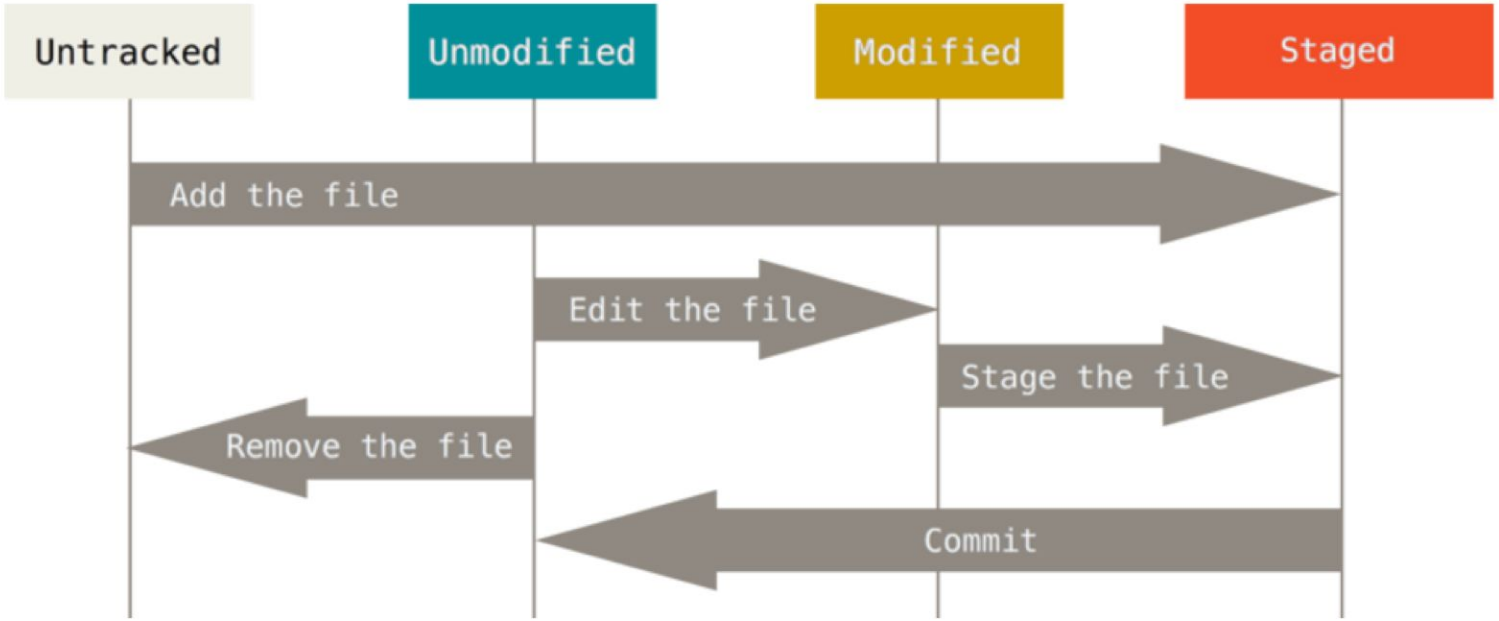
Essential git commands

```
$ # 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 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

The kernel source tree

```
$ git clone https://github.com/torvalds/linux.git # clone the kernel repo
$ cd linux; git checkout v6.15 # checkout v6.15
$ tree -d -L 2  # list top two-level directories
├── arch        # * architecture dependent code
│   ├── arm64   # - 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
│   ├── f2fs    # - 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
646 directories
```

Linux cross reference

- Code indexing tool with a web interface
  - Don’t install it
  - Use it here: https://elixir.bootlin.com/linux/v6.15/source
- Allows to:
  - Browse the code of different Linux kernel code versions
  - Search for identifiers (functions, variables, etc)
  - Quickly lookup a function declaration/definition

Now let’s browse

Build the kernel

1. Configure the kernel
   - Configuration file defining compilation options (~ 3700 for x86)
2. Compile 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

Kernel configuration file: `.config`

- `.config` file is at the root of the kernel source
  - Preprocessor flags in the source code
- Syntax is *KEY=VALUE*, and we have several options to configure the kernel.
- But why we need to configure the kernel?

```
#
CONFIG_CC_VERSION_TEXT="x86_64-linux-gnu-gcc-14
CONFIG_CC_IS_GCC=y
CONFIG_GCC_VERSION=140200
CONFIG_CLANG_VERSION=0
CONFIG_AS_IS_GNU=y
CONFIG_AS_VERSION=24400
CONFIG_LD_IS_BFD=y
CONFIG_LD_VERSION=24400
CONFIG_LLD_VERSION=0
CONFIG_RUSTC_VERSION=108401
CONFIG_RUST_IS_AVAILABLE=y
CONFIG_RUSTC_LLVM_VERSION=190107
CONFIG_CC_CAN_LINK=y
CONFIG_CC_CAN_LINK_STATIC=y
CONFIG_CC_HAS_ASM_GOTO_OUTPUT=y
CONFIG_CC_HAS_ASM_GOTO_TIED_OUTPUT=y
CONFIG_TOOLS_SUPPORT_RELR=y
CONFIG_CC_HAS_ASM_INLINE=y
CONFIG_CC_HAS_NO_PROFILE_FN_ATTR=y
CONFIG_LD_CAN_USE_KEEP_IN_OVERLAY=y
CONFIG_RUSTC_HAS_COERCE_POINTEE=y
CONFIG_PAHOLE_VERSION=129
CONFIG_IRQ_WORK=y
CONFIG_BUILDTIME_TABLE_SORT=y
CONFIG_THREAD_INFO_IN_TASK=y
```

Ref: https://elixir.bootlin.com/linux/v6.15/source/mm/madvise.c#L1423

Configure the kernel

```
make menuconfig
```

- Need libraries and tools
  - **`sudo apt install build-essential libncurses-dev bison flex libssl-dev libelf-dev dwarves fakeroot libdw-dev debhelper-compat`**

Fix the kernel `.config` file

`make menuconfig`

```
    General setup  --->
    Platform selection  --->
    Kernel Features  --->
    Boot options  --->
    Power management options  --->
    CPU Power Management  --->
[*] ACPI (Advanced Configuration and Power Interface) Support  --->
[*] Virtualization  --->
    General architecture-dependent options  --->
[*] Enable loadable module support  --->
[*] Enable the block layer  --->
    Executable file formats  --->
    Memory Management options  --->
[*] Networking support  --->
    Device Drivers  --->
    File systems  --->
    Security options  --->
- - Cryptographic API  --->
    Library routines  --->
    Kernel hacking  --->
```

```
--- Cryptographic API
      Crypto core or helper  --->
      Public-key cryptography  --->
      Block ciphers  --->
      Length-preserving ciphers and modes  --->
      AEAD (authenticated encryption with associated data) ciphers  --->
      Hashes, digests, and MACs  --->
      CRCs (cyclic redundancy checks)  --->
      Compression  --->
      Random number generation  --->
      Userspace interface  --->
      Accelerated Cryptographic Algorithms for CPU (arm64)  --->
[*]   Hardware crypto devices  --->
-*-   Asymmetric (public-key cryptographic) key type  --->
      Certificates for signature checking  --->
< >   Kerberos 5 crypto
```

```
(certs/signing_key.pem) File name or PKCS#11 URI of module signing key
    Type of module signing key to be generated (RSA)  --->
-*- Provide system-wide ring of trusted keys
()    Additional X.509 keys for default system keyring
[ ]   Reserve area for inserting a certificate without recompiling
[ ]   Provide a keyring to which extra trustable keys may be added
[ ] Provide system-wide ring of blacklisted keys
```

Configure the kernel

- **`make defconfig`**
  - Generate the default configuration of the running platform
  - linux/arch/x86/configs/x86_64_defconfig
- **`make oldconfig`**
  - Use the configuration file of the running kernel
  - Will ask about new configurations
    - If unsure, choose the default configurations

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`**

```
sudo apt install build-essential libncurses-dev bison flex
libssl-dev libelf-dev dwarves fakeroot libdw-dev debhelper-compat
```

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 modules_install
$ sudo make install
$ ls /boot/*6.15*
/boot/System.map-6.15.0
/boot/config-6.15.0
/boot/initrd.img-6.15.0
/boot/vmlinux-6.15.0

# Reboot the machine and check if a system boots with the new kernel
$ sudo reboot
$ uname -a
```

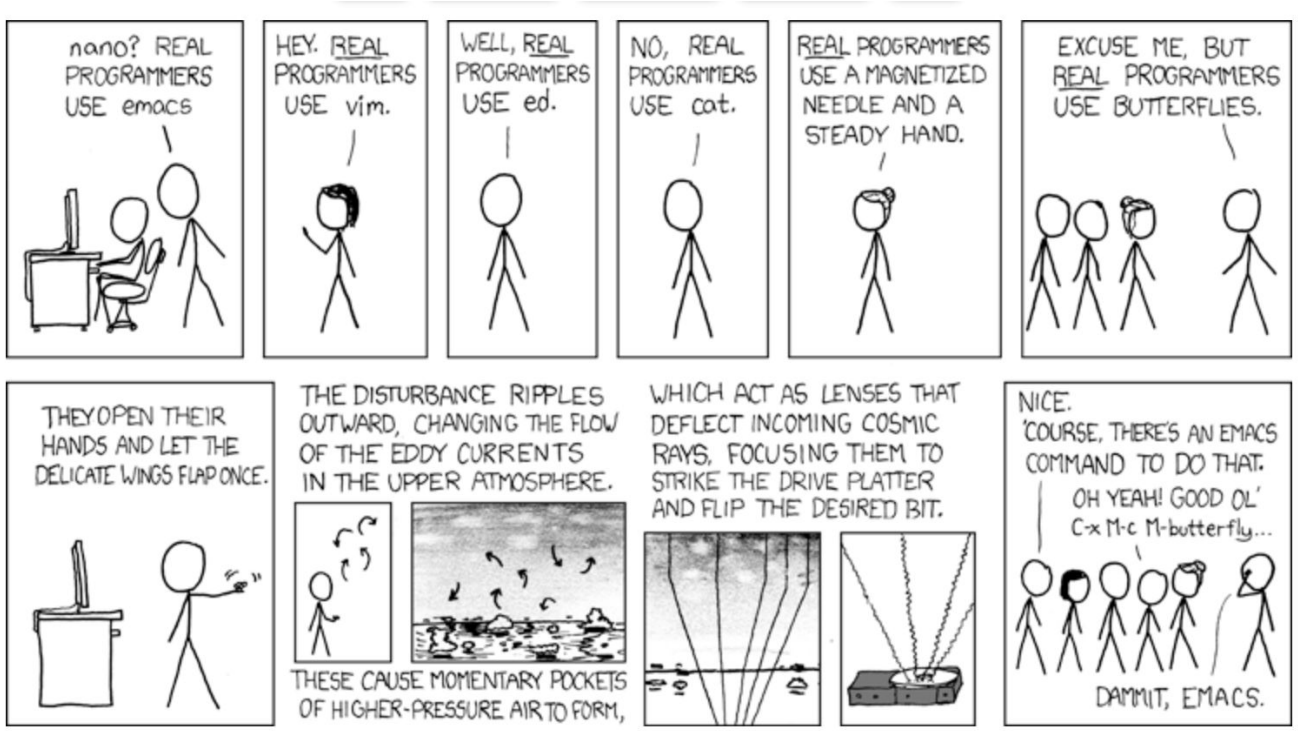
DEMO

What are these files ?

```
/boot/System.map-6.15.0
/boot/config-6.15.0
/boot/vmlinux-6.15.0
/boot/initrd.img-6.15.0
```

Editor

- Several good editors
  - vim, emacs, IDEs (Clion, VSCode)



Source: https://xkcd.com/378

Terminal Multiplexers

- Installation

$ sudo apt install tmux

```
tmux new -t new_session
<C b> + c     # new window
<C b> + x     # kill window
<C b> + d     # detach from session
<C b> + %     # new vertical pane
<C b> + “     # horizantal pane
```

Kernel vs. user programming

- No libc or standard headers
  - Instead, the kernel implements several libc-like functions

- Example:
  - `#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-sized stack: 8KB (2 pages) in X86
- No memory protection
  - SIGSEGV kernel panic (OOPS)
- An example of kernel oops

Kernel vs. user programming

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

Linux kernel coding style

- Indentation 1 tab 8-character (not 8 spaces)
- No **CamelCase,** use **underscores:** `Spinlock` `spin_lock`
- Use C-style comments: `/* use this style */ // not this`

- Line length: 80 characters
- Write code in a similar style to other kernel code

- Ref: Documentation/process/coding-style.rst

Linux kernel coding style

```c
/*
 * 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();
    }
}
```

Other useful resources

- The Linux Kernel Documentation: The extensive documents extracted from the kernel source
- Linux Weekly News: Easy explanation of the recently added kernel features
- Linux Inside: Textbook-style description of kernel subsystems
- Kernel newbies: Useful information for new kernel developers
- Linux Kernel API Manual
- Kernel Recipes
- Kernel planet

Next lecture

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

```
fd = open(“out”, 1);

write(fd, “hello\n”, 6);

pid = fork();
```