Welcome to Linux

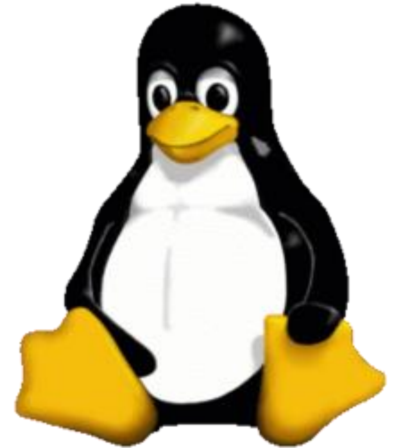
Lecture 1.1

Some history

- 1969 - the Unix operating system by Ken Thompson and Dennis Ritchie
- Unix became widely adopted by academics and businesses
- 1977 - the Berkeley Software Distribution (BSD) by UC Berkeley. A lawsuit *USL v. BSDi*.
- 1983 – the GNU project by Richard Stallman - a free UNIX-like operating system (GPL). GNU incomplete – no kernel
- 1991- Linus Torvalds, an undergraduate student from Finland, began a “just for fun” project that later became the Linux kernel.

Linux operating system

- Open-source
- Written in portable yet highly efficient language
- Built-in networking
- Built-in multitasking
- Rich software development environment
- Open interface to kernel
- Powerful and flexible CLI (command-line interface)



“UNIX is very simple, it just needs a genius to understand its simplicity”

Dennis Ritchie,
creator of C programming language

Scope for the first 2 weeks

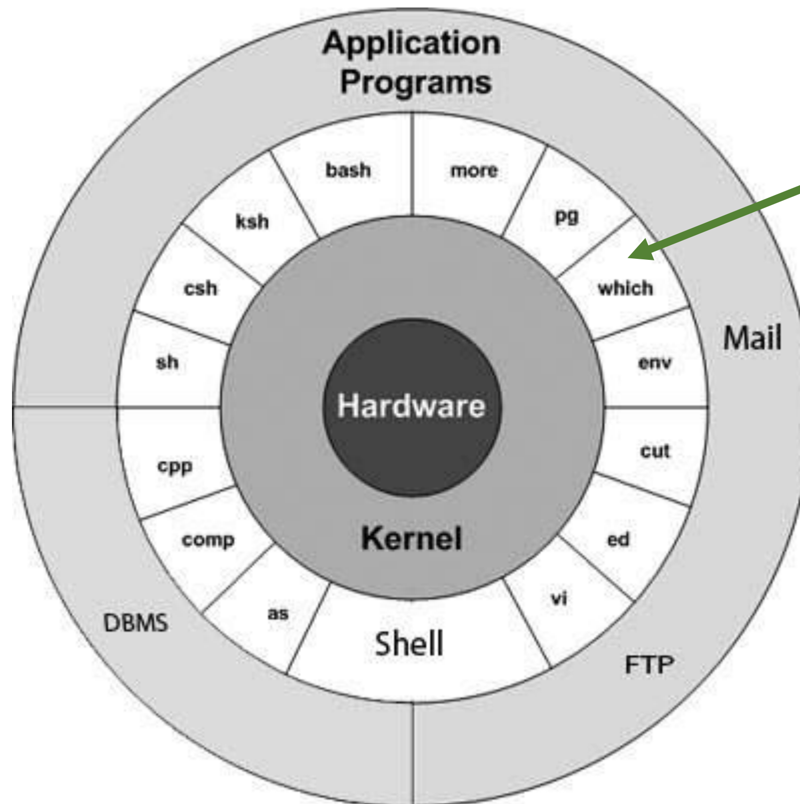
- Get familiar with Linux
- Use existing utilities with CLI
- Shell programming

Scope for the rest of the course

- Develop our own utilities (tools) in a Unix-style
- Write application programs which interact with Linux kernel

- All this using C programming language

Linux structure



Tools: We are interested in these

Linux kernel

- Process creation, and scheduling multiple processes
- Memory management: allocation, release
- File system on disk: abstraction over physical disk blocks
- Access to I/O devices: device drivers
- Networking: routing and exchange of messages
- Interface for user programs to perform requests to kernel: system calls

Linux file system

```
ls -li
```

File abstraction

- “Everything is a file.”
- **Unified** file interface = open, read, write, close for
  - regular files
  - directories
  - devices
    - video
    - keyboard
    - network

Index node - inode

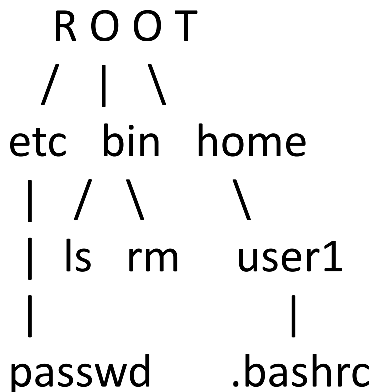
- The data for each file is managed by an array of on-disk data structures called *inodes*
- One inode is allocated for each file and each directory
- Unix inodes have unique numbers, not names, and it is these numbers that are kept in directories alongside the names.

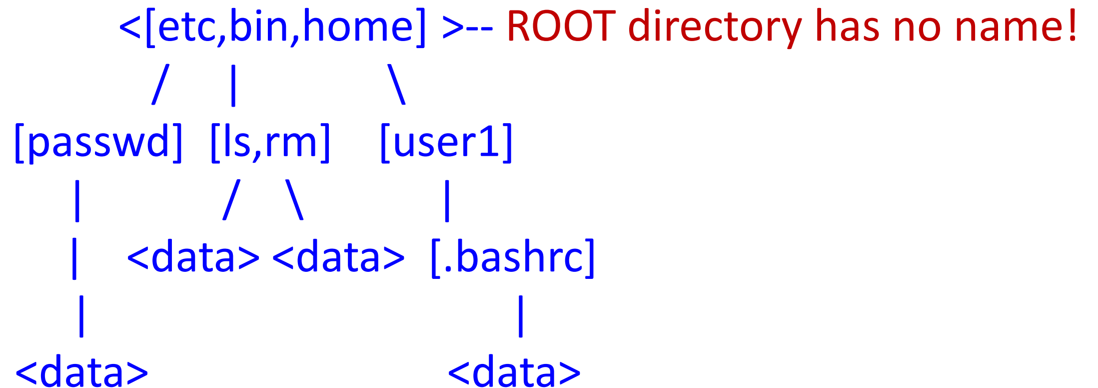
`ls -i`

Typical Linux file hierarchy

- Everything starts in the "root" directory
- A directory is a file that contains directory entries:

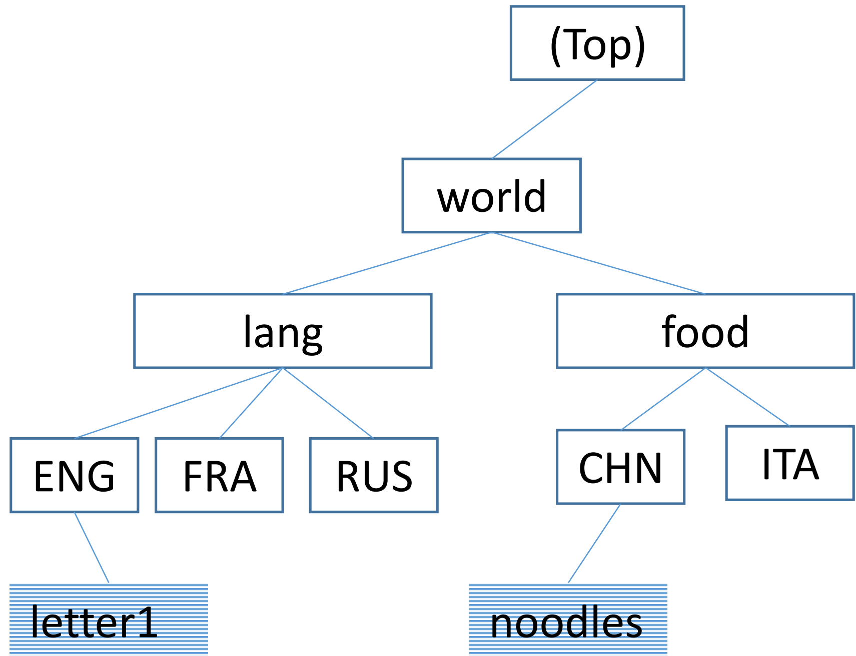
  pairs of (child name, inode).





What is stored in inodes - example

i	directory	What is stored
8	top	[world-9, …]
9	world	[lang-10, food-11]
10	lang	[ENG-12, FRA-16, RUS-17]
11	food	[CHN-13, ITA-18]
12	ENG	[letter1-14]
13	CHN	[noodles-15]
14	letter1	File data blocks
15	noodles	File data blocks



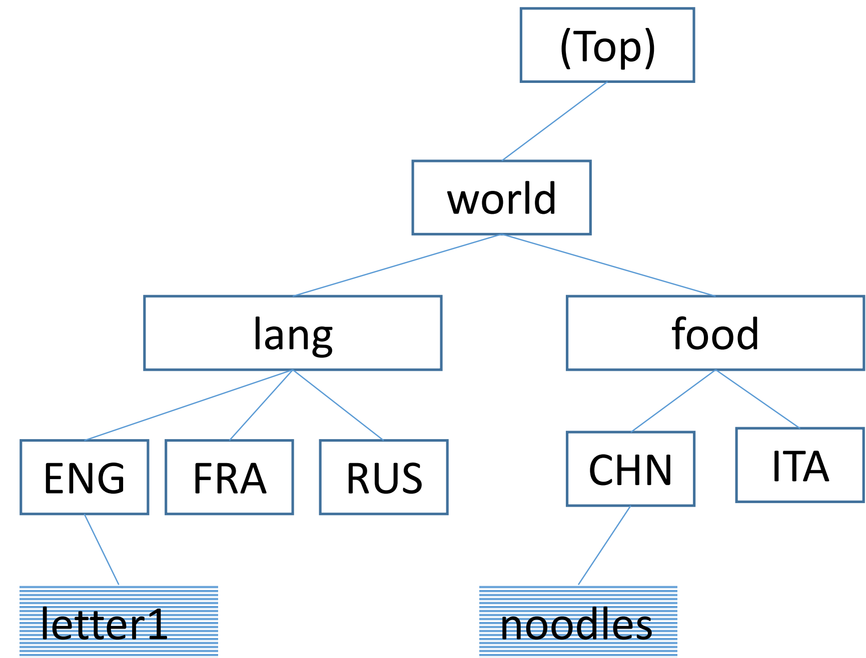
File vs. directory inodes

- File inode – location of data
- Directory inode – location of (name, inode) pairs for child directories

- You must use the inode number from the directory to find the inode on disk to read its attribute information; reading the directory only tells you the name and inode number.

What is NOT stored in inodes?

i	directory	What is stored
8	top	[world-9, …]
9	world	[lang-10, food-11]
10	lang	[ENG-12, FRA-16, RUS-17]
11	food	[CHN-13, ITA-18]
12	ENG	[letter1-14]
13	CHN	[noodles-15]
14	letter1	File data blocks
15	noodles	File data blocks



The name of a file is NOT stored in file inode – it is stored in the parent directory

Noname files

- The name and inode number pair in a directory is the only connection between a name and the thing it names on disk
- If a directory is damaged, the names of the things are lost and inodes become “orphan”
- The things themselves may be undamaged. You can run a file system recovery program such as `fsck` to recover the data (but not the names)

What else is stored in inodes

In addition to a list of pointers to the disk blocks:

- The attributes of the file or directory itself (permissions, size, access/modify times, etc.); but, not the name of the file or directory:

  The names are kept separately, in parent directories

- Directory inode stores two additional (name, inode) pairs:

  Itself: . → inode

  Parent: .. → inode

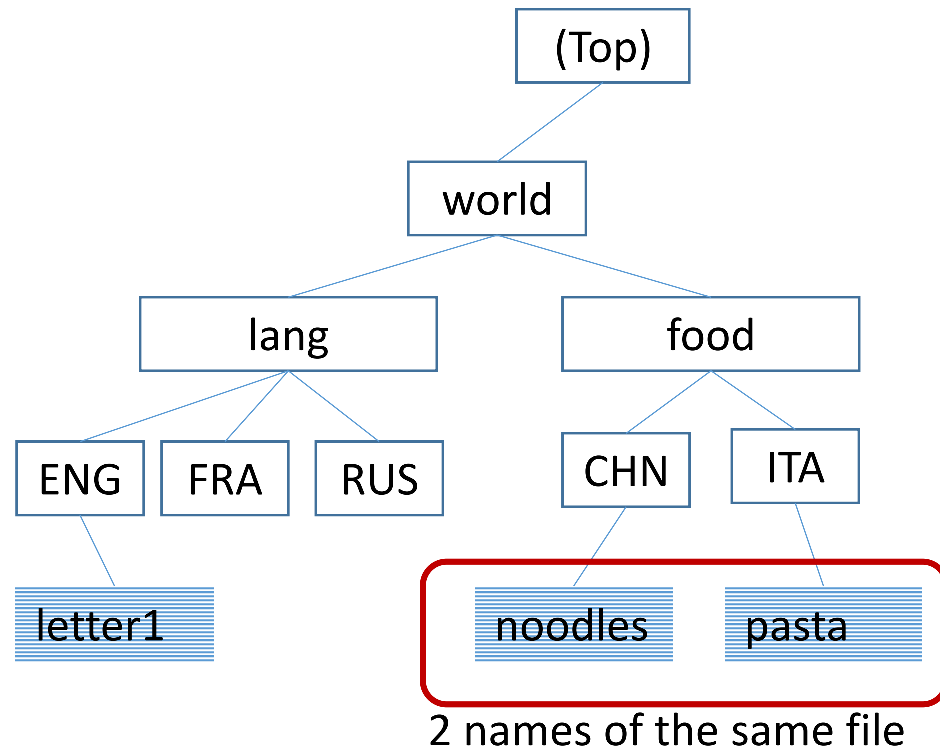
Multiple names to the same file: hard links

- An entry in a directory file which specifies a pair of (name, inode) is called a hard link.
- There can be several hard links to the same physical file!

```
ln bar foo
ls -li
```

Hard link example

i	directory	What is stored
11	food	[CHN-13, ITA-18]
12	ENG	[letter1-14]
13	CHN	[**noodles-15**]
14	letter1	File data blocks
15	noodles	File data blocks
16		…
17		…
18	ITA	[**pasta-15**]




```
cd world/food
ln CHN/noodles ITA/pasta
```

Tracing inodes example: /home/alex/foobar

```
    +----+-----+-----------------------------------------+
#2  |. 2 |.. 2 | home 5 | usr 9 | tmp 11 | etc 23 | ...  |
    +----+-----+-----------------------------------------+
                 |  The inode #2 above is the ROOT directory. It has the
                 |  name "home" in it. The *directory* "home" is not
                 |  here; only the *name* is here. The ROOT directory
                 |  itself does not have a name!
                 V
    +----+-----+-----------------------------------------------------+
#5  |. 5 |.. 2 | alex 31 | leslie 36 | pat 39 | abcd0001 21 | ...   |
    +----+-----+-----------------------------------------------------+
                 |  The inode #5 above is the "home" directory. The name
                 |  "home" isn't here; it's up in the ROOT directory,
                 |  above. This directory has the name "alex" in it.
                 V
    +----+-----+-----------------------------------------------------+
#31 |. 31|.. 5 | foobar 12 | temp 15 | literature 7 | demo 6 | ... |
    +----+-----+-----------------------------------------------------+
                 |  The inode #31 above is     |
                 |  the "alex" directory. The  |
                 |  name "alex" isn't here;    |
                 |  it's up in the "home"      |
                 |  directory, above.  This    |
                 |  directory has the names    |
                 |  "foobar" and "literature"  |
                 |  in it.                     |
                 |                             V
    +----+-----+--|------------------------------------------+
#7  |. 7 |.. 31|  |  barfoo 12 | morestuf 123 | junk 99 | ... |
    +----+-----+--|------------------------------------------+
                 |       |  The inode #7 above is the "literature" directory.
                 |       |  The name "literature" isn't here; it's up
                 |       |  in the "alex" directory.  This directory has
                 |       |  the name "barfoo" in it.
                 |       |
                 V       V
                *-----------*  This inode #12 on the left is a file inode.
                | file data |  It contains the data blocks for the file.
            #12 | file data |  This file happens to have two names, "foobar"
                | file data |  and "barfoo", but those names are not here.
                *-----------*  The names of this file are up in the two
                               directories that point to this file, above.
```

From: http://teaching.idallen.com/cst8207/13w/notes/450_file_system.html

Directories cannot have hard links!

- Files may have many names ("links") - but directories can not!
- Each directory inode is allowed to appear once in exactly one parent directory and no more.
- Every sub-directory only has one parent directory, and the special name ".." (dot dot) always refers unambiguously to its unique parent directory
- This directory linking restriction prevents loops and cycles in the file system tree

ln vs. ln –s

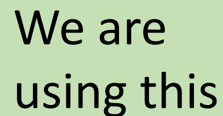
- Storage Space: no new inodes with hard links - in soft links we create a new inode to store the path to the file
- Performance: directly accessing the disk pointer instead of going through the path stored in soft link file.
- Renaming (mv) target file: the hard link will still work, but soft link will point to the previous file location.
- Redundancy: with hard link, the data is safe, until all the links to the file are deleted - in soft link, you will lose the data if the master instance of the file is deleted.

Programmable shell

Running built-in utilities

Shells

- Special-purpose programs designed to read commands typed by the user and shell scripts, interpret them, and execute appropriate programs in response
- Many shells, i.e.:
  - Bourne shell (SH)
  - Bourne again shell (BASH)



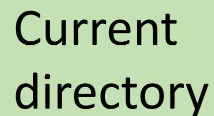
How the shell is collaborating with the kernel

- Shell:
  - accepts command names and arguments as input
  - finds the executable
  - interprets the arguments
  - loads an executable into memory and hands it off to the OS to run.
- Kernel:
  - starts the process of executing the program

How does shell know where to find an executable

- PATH variable: List of directories to be consulted when looking up commands specified without path names.
- E.g. you type "cat", it execs "/bin/cat". It finds it by looking through the path, which is a list of directories including /bin.

echo "$PATH"



/bin:/usr/bin:.

PATH=$PATH:/path/to/dir1; export PATH

To add permanently:

echo 'export PATH=$PATH:/usr/local/bin' >> ~/.bash_profile

Globbing

- Globbing - process of expanding a non-specific file name containing a wildcard character into a set of specific file names that exist
- Standard wildcards (globbing patterns)
  - * matches any number of any character
  - ? matches any one character
  - [range] :
    - `m[a,o,u]m, m[a-d]m`
  - {} matches at least one (or):
    - `cp {*.doc,*.pdf} ~`
  - [!] excluding
    - `rm myfile[!9]`

Sharing files: permissions

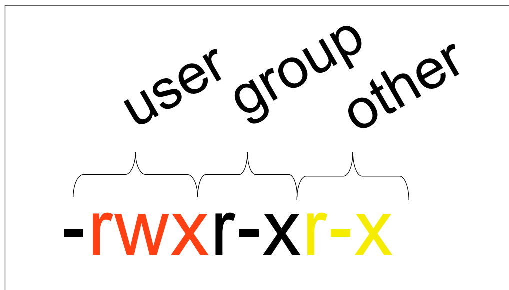
Users belong to user groups (up to 16-32 groups max)

wolf:~% groups mgbarsky

mgbarsky : instrs csc209h csc343h csc443h cs209hi cs343hi cs443hi

Permissions as numbers

Number	Octal Representation	Ref
0	No permission	---
1	Execute permission	--x
2	Write permission	-w-
3	Execute and write permission: 1 (execute) + 2 (write) = 3	-wx
4	Read permission	r--
5	Read and execute permission: 4 (read) + 1 (execute) = 5	r-x
6	Read and write permission: 4 (read) + 2 (write) = 6	rw-
7	All permissions: 4 (read) + 2 (write) + 1 (execute) = 7	rwx



Setting permissions

- chmod 755 <filename>
  - 3 numbers between 0 and 7, the octal value for that category of user
  - Quiz — what is the command to set the permissions of the file *classlist* to be world readable but writeable only by the file owner and members of the group.
- Or using:
  - chmod u+rwx
  - chmod go-x
  - chmode a=x
  - adds or removes permissions for those categories of users

File Permissions

chmod (change mode)

- Changes the permissions (mode) on an existing inode (file, directory, etc.)

ls -lid (list structure, long version, inode, directory)

- Shows the permissions of an inode

Output redirection

- If the notation `> file` is appended to any command that normally writes its output to standard output, the output of that command will be written to file:

```
who > users
```

Input redirection

- The commands that normally take their input from standard input can have their input redirected from a file:

```
wc -l users
wc -l < users
```

Processes

Kernel starts a process for each program

To see all the processes:

ps

```
 PID TTY          TIME CMD
26357 pts/5   00:00:00 tcsh
26558 pts/5   00:00:00 bash
32624 pts/5   00:00:00 ps
```

Process groups and pipelining

- Connect processes, by letting the standard output of one process feed into the standard input of another. That mechanism is called a *pipe*.
- Connecting simple processes in a pipeline allows to perform complex tasks without complex programs.

```
$ls -l | sort -k5n | less
```

Displays files in current directory sorted by file size

grep

- Searching plain-text data sets for lines matching a regular expression.
- Main uses:
  - grep –x matches entire line
  - grep –v matches all lines which do not contain a pattern
  - grep ^pattern – matches lines which start with ‘pattern’

Summary: your Linux toolbox

- Linux file system: inodes, hard and soft links
- File permissions
- Working with files
- Working with file contents
- I/O redirection
- Pipelining