Implementing case-insensitive file systems in the Linux kernel

André Almeida

andrealmeid (at) igalia.com / tonyk (IRC/Matrix)

Open Source Summit Japan 2025

Talk summary

- What are letter cases anyway?
- Where case-insensitive file systems comes from?
- What's the use cases for Linux?
- What's so bad about it?

Letter cases

- A minority of writing systems have letter cases (bicameral script)

Aa Bb Cc

Aα Bβ Γγ

Гг Дд Ии

Letter cases

- But most doesn’t

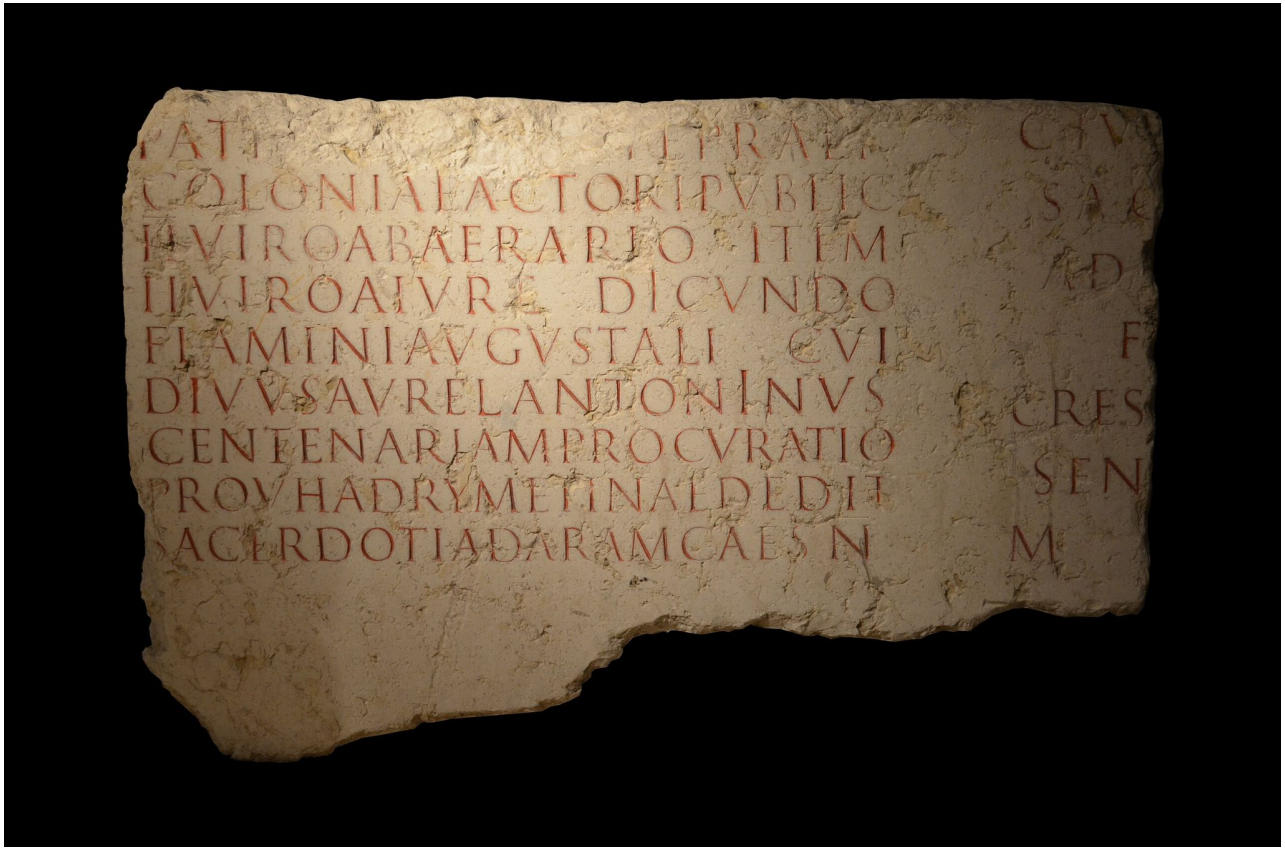
作家、科學家

يولد جميع
الناسراً

모든 인간은

すべての人間

Letter cases



Letter cases

Morse code

- Binary system
- Less resolution is easier to operate

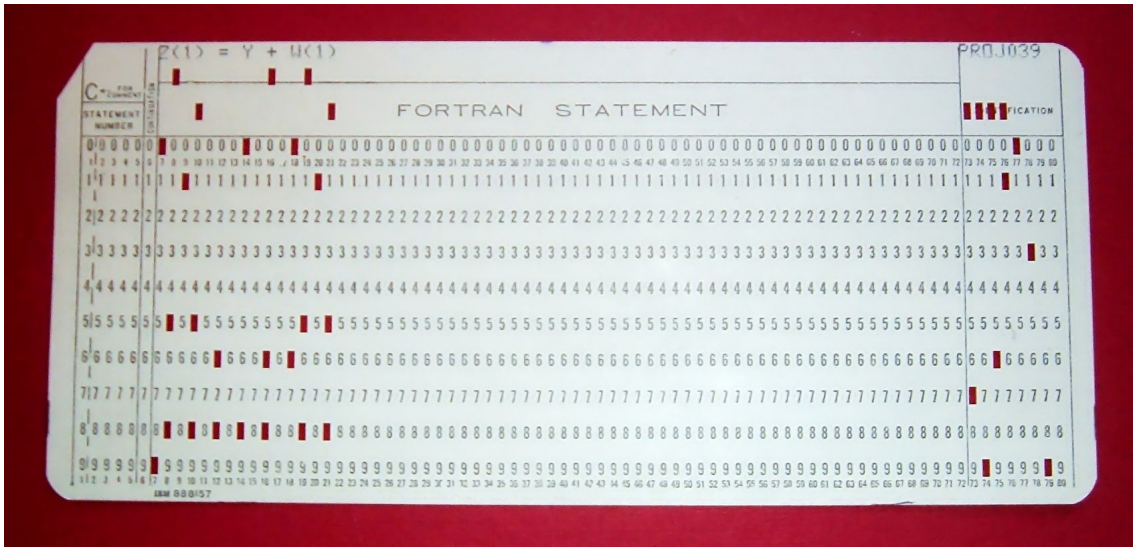
```
      .-.. .. -. ..- -..-
.--. .-.. ..- -- -... . .-. ...
```

LINUX PLUMBERS

The computer and the letter cases

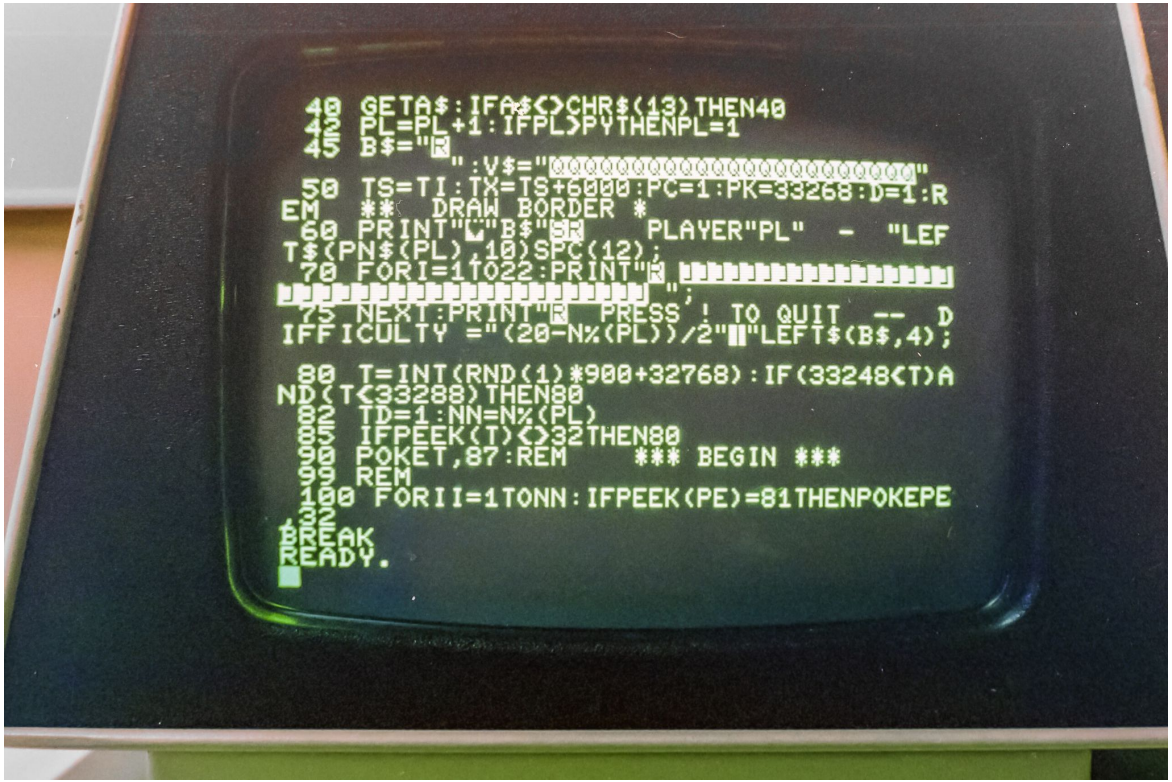
Programing with punched cards

- Not enough resolution to fit the alphabet twice
- “Early IBM computers did not support lowercase letters…”
- “The standard Algol Elliott 803 used five-hole paper tape and thus only had upper case.”



All caps programming

- FORTRAN, ALGOL, BASIC



All caps programming

- Computers would run a single application, thus they would keep track of the file locations themselves
- With multitasking, OSs had to keep record of files with the first file systems

All caps programming

- Computers would run a single application, thus they would keep track of the file locations themselves
- With multitasking, OSs had to keep record of files with the first file systems
- Some OSes, like OS/360 and MS-DOS (FAT) had all caps file systems
- Even if the user was able to create a new file mixing letter cases, the final filename would be converted to uppercase (is not a **case preserving** file system)

All caps programming

- Eventually, FAT started to support lowercase file names, but since it had a previous behaviour of converting names to uppercase, to be backward compatible with the software it had to be **case-insensitive**

All caps programming

- Eventually, FAT started to support lowercase file names, but since it had a previous behaviour of converting names to uppercase, to be backward compatible with the software it had to be **case-insensitive**
- After that, the next Windows file systems (like NTFS) had to be case-insensitive as well
- And it will be like that forever to keep the backward compatibility!
- MacOS and Android's user fs are **case-insensitive** by default

All caps programming

- Is it user friendly?
  - `Contract.pdf` vs `contract.pdf`
  - tab completion in bash:
    - `vim .Xauthorrity`
    - `sudo systemctl status NetworkManager`
- The machines are case-sensitive, but we are not!

Unix file system

- Unix was created from scratch, not much longer after the ASCII table was created, and had no backward compatibility to care about
- The first Unix file system was **case-sensitive**

Unix file system

- Unix was created from scratch, not much longer after the ASCII table was created, and had no backward compatibility to care about
- The first Unix file system was **case-sensitive**
- It's much easier for the computer (and to the programmer) to deal with case-sensitive string comparison
- The relation between *A* and *a* is easy for humans, but for the computer they are different numbers
- Linux, based on Unix, followed the same path

Do we need case-insensitive on Linux?

Windows games

- Windows game developers correctly expects that their platform will be a **case-insensitive** one
- They can write `open("IMAGE.PNG")` or `open("image.png")` and both are correct

Windows games

- Windows game developers correctly expects that their platform will be a **case-insensitive** one
- They can write `open("IMAGE.PNG")` or `open("image.png")` and both are correct
- NTFS is **case-preserving** and guarantee one file per name, regardless of the letter case
  - Cannot create file `DOG.jpg`, because `dog.jpg` already exists

Windows games

- Installing a Windows game on ext4, and those expectations aren't true anymore

Windows games

- Installing a Windows game on ext4, and those expectations aren't true anymore
- If a file is not found, it might be a case-sensitive issue. Retry again for every file in the directory making a case-insensitive string compare
- Wine (the Windows compatibility tool) needs to "scan" the filesystem
- Both file names are normalized (i.e. casefolded) to a common case type (like lowercase) and then compared

Windows games

```
games/CoolGame/assets/Images/characters/Enemies/ENEMY1.jpg
```

Windows games

- Doing this in userspace has two major issues
- Performance: for every part of the file path, Wine needs to ask the filesystem for every file there, casefold, strcmp(), etc
  - The Kernel VFS already have everything in place for searching filenames in a fast way

Windows games

- Doing this in userspace has two major issues
- Performance: for every part of the file path, Wine needs to ask the filesystem for every file there, casefold, strcmp(), etc
  - The Kernel VFS already have everything in place for searching filenames in a fast way
- Correctness: if Wine founds two files with the same name but different cases, which one should be opened?
  - Only a filesystem approach can prevent this to happen

Windows games

- Modding tools are custom software made by gaming community to change some parts of the game, like changing textures, adding new game modes
- Modders also expects that the game will be running in Windows and a lot of issues appeared when using them on Linux (creating duplicate files, inconsistently applying changes, etc)

So let’s implement on Linux kernel

Case-insensitive and Linux

- Back on the FAT days, the main encoding was the ASCII table
- Very easy to casefold, but very limited

Case-insensitive and Linux

- Back on the FAT days, the main encoding was the ASCII table
- Very easy to casefold, but very limited
- English is the only language that can fit in the ASCII table (if you exclude words like *résumé*)
- Every other language that uses Latin alphabet has at least one diacritic (e.g. é, ñ, ç), and they casefold as well
- CAFÉ -> café
- NTFS supports this using **Unicode!**

Case-insensitive and Linux

- Unicode have this huge encoding tables (UTF-8) aiming to support all scripts and special characters
- It also set the rules for **normalization** and **casefolding**, with a big mapping table

Normalization

- There are many ways to write the same character in Unicode
- When the user is looking for a string, they don't care about this, so for string compares we need to have them in a same standard form
- Without normalization, a filesystem can have the same file names multiple times: there are 4 ways to write "coração"

Normalization

- **Ç**
  - **Ç** [U+00C7 LATIN CAPITAL LETTER C WITH CEDILLA]
  - **C** [U+0043 LATIN CAPITAL LETTER C] followed by ◌̧ [U+0327 COMBINING CEDILLA]

EXAMPLE 12: Encoding Variations

Consider the character Ǻ [U+01FA LATIN CAPITAL LETTER A WITH RING ABOVE AND ACUTE]. One way to encode this character is as U+01FA LATIN CAPITAL LETTER A WITH RING ABOVE AND ACUTE. Here are some of the different character sequences that a document could use to represent this character:

Ǻ — U+01FA —A "precomposed" character.

Ǻ — A + U+030A + U+0301 — A 'base' letter 'A' followed by two combining marks (U+030A COMBINING RING ABOVE and U+0301 COMBINING ACUTE ACCENT)

Ǻ — U+00C5 + U+0301 —An accented letter (U+00C5 LATIN CAPITAL LETTER A WITH RING ABOVE) followed by a combining accent (U+0301 COMBINING ACUTE ACCENT)

Ǻ — U+212B + U+0301 —A compatibility character (U+212B ANGSTROM SIGN) followed by a combining accent (U+0301 COMBINING ACUTE ACCENT)

Ａ́ — U+FF21 + U+030A + U+0301 — A compatibility character U+FF21 FULLWIDTH LATIN LETTER CAPITAL A) followed by two combining marks (U+030A COMBINING RING ABOVE and U+0301 COMBINING ACUTE ACCENT)

Each of the above strings contains the same apparent "meaning" as Ǻ [U+01FA LATIN CAPITAL LETTER A WITH RING ABOVE AND ACUTE], but each one is encoded slightly differently. More variations are possible, but are omitted for brevity.

Normalization

- Of course, there are 4 ways to normalize strings
- As normalization can change the meaning of things, some algorithms tries to keep the original meaning even after normalizing
- The one choose for Linux kernel was the **NFD**:
  - Characters are decomposed by canonical equivalence, and multiple combining characters are arranged in a specific order.

Casefolding

- The casefold mapping file defines a “standard” case for each character
- `00C9; C; 00E9; # LATIN CAPITAL LETTER E WITH ACUTE`
  - `U+00C9 -> É`
  - `U+00E9 -> é`
- If a characters is not found in the table, it just folds to itself

Casefolding

- Fun examples:
  - Eszett (or sharp S) ß is a German letter that had no formal uppercase form
  - It’s transformed into SS, changing the length of the string!
  - straße -> STRASSE
  - Today that’s a “ß LATIN CAPITAL LETTER SHARP S” that is also accepted

Casefolding

- Fun examples:
  - Greek Sigma has three shapes:
  - Σ GREEK CAPITAL LETTER SIGMA
  - σ GREEK SMALL LETTER SIGMA
  - ς GREEK SMALL LETTER FINAL SIGMA
  - They all casefold to σ

Casefolding + Normalization

EXAMPLE 19

Original		Case Fold		NFKC		Case Fold		NFKC
㎒	=>	㎒	=>	MHz	=>	mhz	=>	mhz
U+3392		U+3392		U+004D U+0048 U+007A		U+006D U+0068 U+007A		U+006D U+0068 U+007A
°Ć	=>	°Ć	=>	°Ć	=>	°ć	=>	°ć
U+2103 U+0301		U+2103 U+0301		U+00B0 U+0106		U+00B0 U+0107		U+00B0 U+0107
Ϊ́	=>	ΐ	=>	ΐ	=>	ΐ	=>	ΐ
U+03AA U+0301		U+03CA U+0301		U+0390		U+03B9 U+0308 U+0301		U+390

Unicode on Linux

- **Gabriel Krisman** finally merged Unicode support for Linux filesystems in v5.2
  - The name of file is still an opaque sequence of bytes, to be correctly displayed you need a userspace terminal/file manager with Unicode support

Unicode on Linux

- **Gabriel Krisman** finally merged Unicode support for Linux filesystems in v5.2
  - The name of file is still an opaque sequence of bytes, to be correctly displayed you need a userspace terminal/file manager with Unicode support
- The kernel has a library that fs devs can use to enable encoding support for a filesystem, and casefold is a bonus!
- It’s not integrated on the VFS level, so it’s not enabled for every filesystem

Unicode on Linux

- `struct dentry_operations generic_ci_dentry_ops`
- Before calling `d_hash()` and `d_compare()`, a canonical string is formatted with normalization and casefolding

Unicode on Linux

- `struct dentry_operations generic_ci_dentry_ops`
- Before calling `d_hash()` and `d_compare()`, a canonical string is formatted with normalization and casefolding
- It's **disabled by default!** User needs to explicitly ask to a directory to be case-insensitive. All child directories will follow.
  - Only works on empty directories to avoid problems
- Currently supported by ext4, f2fs, tmpfs and overlayfs
  - I have added support for the last two to enable game containers to work with casefold mount points

Unicode on Linux

- overlayfs is a meta filesystem used to merge two different mount points in a single one, using layers
- If a file exists in both layers, just the upper layer version is presented. Directories with the same name are merged.

Unicode on Linux

- overlayfs is a meta filesystem used to merge two different mount points in a single one, using layers
- If a file exists in both layers, just the upper layer version is presented. Directories with the same name are merged.
- To have casefold support, overlayfs now stores the files name twice: one to be used for comparing and merging, another one to be used to be displayed to the user
- We honor the case that was used in the upper layer (**case preserving**)

What’s bad with case-insensitive?

- Case-insenstive doesn’t have a very good reputation :)
- People on the internet get really angry about this topic, in forums and LKML

What’s bad with case-insensitive?

- It’s hard to get it right:
  - Each normalization algorithm has its own drawbacks
  - Changes made to the Unicode mapping or the casefold algorithm can make files unreachable
    - `Revert "unicode: Don't special case ignorable code points"`

What's bad with case-insensitive?

- It's hard to get it right:
  - Each normalization algorithm has its own drawbacks
  - No support for negative dentries right now
  - Security issues?
  - Changes made to the Unicode mapping or the casefold algorithm can make files unreachable
    - `Revert "unicode: Don't special case ignorable code points"`
  -

The Turkish i

- To be 100% precise, casefolding needslocalization
- English
  - I -> i
- Turkic languages:
  - I -> ı
  - İ -> i
- IGALIA -> igalia or ıgalıa?
- igalia -> IGALIA or İGALİA?

Join us!

https://www.igalia.com/jobs

Image credits:

Wikimedia

https://www.w3.org/TR/charmod-norm/#definitionCaseFolding