

M set to the time of the operation **m** kept from the source file **M** set to a value the command chose · unchanged **A^{r s}** only under these mount options: **r** reltime (default), **s** strictatime, **n** noatime. No letters: all three **NEW** new inode: B is fresh

OPERATION	FILE				PARENT FOLDER				DESTINATION FOLDER				READ IT AS
	M	A	C	B	M	A	C	B	M	A	C	B	
CREATE AND CHANGE CONTENT													
Create a file touch f, > f	M	A	C	B	M	.	C	.					NEW New inode, all four equal.
Append or overwrite in place echo x >> f	M	.	C	.	.	.	.	.					Same inode. Folder untouched.
Save in nano nano f	M	A ^{r s}	C	.	.	.	.	.					In place. A from nano's read: r once after a change, then daily; s always.
Save in vim vim f (shipped defaults)	M	A	C	B	M	.	C	.					NEW New file renamed over the old one: B is the save time.
Edit with sed -i sed -i s/a/b/ f	M	A	C	B	M	.	C	.					NEW Same as vim: a new inode replaces the old.
READ AND RUN													
Read, first time since a write cat f	.	A ^{r s}	.	.	.	.	.	.					r: first read after a change writes A, then at most once a day. s: every read.
Read again the same day cat f	.	A ^s	.	.	.	.	.	.					s: every read writes A. r: skipped until A is a day old or the file changes.
Execute a script or binary ./f	.	A ^{r s}	.	.	.	.	.	.					Same as a read: r once after a change, then daily; s every time.
List a folder ls d					.	A ^{r s}	.	.					Folder A, same as a read: r once after a change, then daily; s every time.
CHANGE THE INODE													
Change permissions or owner chmod, chown	.	.	C	.	.	.	.	.					C only. On the file, the same signature as a rename.
Hard link ln f d2/g	.	.	C	.	.	.	.	.	M	.	C	.	Link count changed, so C moves on the file.
Symbolic link ln -s ../d1/f d2/l	.	.	.	.	.	.	.	.	M	.	C	.	Target untouched. The link is a new inode in d2.
MOVE AND COPY													
Rename in the same folder mv f g	.	.	C	.	M	.	C	.					Same inode. Only C moves on the file.
Move to another folder, same file system mv f ../d2/	.	.	C	.	M	.	C	.	M	.	C	.	Same inode. Both folders change.
Move to another file system mv f /other/	m	a	C	B	M	.	C	.	M	.	C	.	NEW Copy, then delete. M and A kept to the nanosecond: B later than M.
Copy cp f ../d2/	M	A	C	B					M	.	C	.	NEW Source A moves on ext4 (r s). On XFS never: cp clones the blocks.
Copy keeping times cp -p, cp -a	m	a	C	B					M	.	C	.	NEW M and A kept to the nanosecond. B later than M.
Copy in a GNOME file manager gio copy -p	m	a	C	B					M	.	C	.	NEW GLib before 2.74 (RHEL 9) cuts M and A to microseconds.
Extract from tar tar -xf a.tar	M	A	C	B					M	.	C	.	NEW M from the archive, whole seconds. A, C, B: time of extraction.
SET TIMES (TIMESTAMPING)													
Backdate M touch -m -d 2020-01-01 f	M	.	C	.	.	.	.	.					C after M: looks like any edited file. Nanoseconds of M are zero.
Date forward touch -d 2030-01-01 f	M	A	C	.	.	.	.	.					M and A set, C now. C earlier than M is the tell.
Backdate A touch -a -d 2020-01-01 f	.	A	C	.	.	.	.	.					C moves, so a set A always shows on C.
DELETE													
Delete a file rm f					M	.	C	.					Folder M and C move. The inode is freed.

The four times

- M** Content last changed. ext4 i_mtime, XFS di_mtime.
- A** Content last read, subject to the mount option. Most reads on a default system leave no trace.
- C** Inode last changed: content, permissions, owner, links, name or another timestamp. Not creation.
- B** Inode created. ext4 i_crtime (256-byte inodes), XFS di_crtime (v5). Not in POSIX.

No Linux interface sets C or B to a chosen value; setting A or M moves C to now. FreeBSD and macOS can set the birth time.

Access time under each mount option

- relatime** Default. A is written on the first read after a change (old A not later than M or C), then at most once every 24 hours.
- strictatime** Every read writes A, every time.
- noatime** Reads never write A. touch -a, cp -p and new files still set it.
- lazytime** Times change in memory first; the disk lagged until unmount. A crashed host's image can hold older times.

Live: findmnt -no TARGET,OPTIONS. Image: /etc/fstab, systemd.mount units, rootflags= on the kernel command line. None set: reltime.

Resolution, range and ticks

- ext4, 256-byte inode:** nanoseconds in an extra field (upper 30 bits); low 2 bits extend the range to 2446.
- ext4, 128-byte inode:** whole seconds, no B, ends in 2038.
- XFS v5 with bigtime** (RHEL 9, 10, Ubuntu 24.04): nanoseconds, B present, range to 2486.
- Clock tick:** steps inside one tick tie to the nanosecond; gaps come in whole ticks. A tie says nothing about order.

KERNEL	TICK	3 COMMANDS	TIED
5.14 (RHEL 9.x), 6.8 (Ubuntu 24.04)	1 ms		0 of 30
6.12 (RHEL 10.x)	1 ms	3 to 7 of 30	
6.18 (WSL 2)	4 ms	20 to 24 of 30	

Tells, and the boring explanation

- C earlier than M:** M set into the future. Or a clock correction: thousands of files is a clock event.
- Whole-second M among nanosecond neighbors:** a typed date, or a file from tar or zip.
- B later than M:** a move across file systems or a copy that keeps times.
- Fresh B on an old file:** an editor that writes a new file (vim, sed -i). B is the age of the inode.
- M and A ending in 000 ns:** a GNOME copy on GLib before 2.74. A lead, never proof.

Read the inode, not the view

```
debugfs -R 'stat <12>' /dev/sdb1
xfs_db -r -c 'inode 133' \
-c 'print core.mtime core.atime core.ctime v3.crtime' /dev/sdc1
dumpe2fs -h /dev/sdb1 | grep 'Inode size'
xfs_info /mnt | grep -E 'isize|bigtime|reflink'
```

ext4 prints seconds, then the extra field: nanoseconds = extra >> 2. stat shows the kernel's copy in memory.

Method. Each operation ran on a fresh loop image: setup, unmount, read the inode, mount, wait past a clock tick, run the operation, unmount, read again. Every value comes from debugfs or xfs_db on the unmounted image, never from stat. ext4 and XFS each measured under reltime, strictatime and noatime.

Measured on

Kernel 5.14
Kernel 6.8
Kernel 6.12
Kernel 6.18
RHEL 9.4 (VM); every RHEL 9 ships 5.14. GLib 2.68
Ubuntu 24.04 (VM), standard kernel. GLib 2.80
AlmaLinux 10.2 (VM), the RHEL 10 kernel. GLib 2.80
WSL 2 kernel, Ubuntu 24.04 and RHEL 10 on top. GLib 2.80

Builds on Thierry and Müller, A systematic approach to understanding MACB timestamps on Unix-like systems, DFRWS EU 2022 (ext4). Every operation both tested agrees. Added here: XFS, current kernels and tools, mount options, copy tools and clock ticks. Full write-up: alexsta-cybersecurity.com/field-notes/reading-linux-timestamps