

LTVM: tiny VMs, LLM tools, and their updates

Marc-André Vef – DDN

Patrick Farrell – The Lustre Collective

LAD 2026 – Developer Summit 26

Overview

- By the end of this presentation, you will know:
 - What Itvm is
 - How to use Itvm to easily cross-build Lustre for different distros and architectures
 - E.g., Rocky 9 host, Ubuntu/Rocky 8 VMs
 - How to create an Itvm VM in ~30 seconds with Lustre mounted
 - How to start/stop/crashdump VMs
 - How a set of CLI tools lets an LLM agent **use** all of this on your behalf
 - review code, triage CI, analyse vmcores
 - What changed since LUG 26 in Indianapolis
- Let's start with Itvm

Developer Friction – Testing and Building

- Lustre is a large project, and an old project...
- Building Lustre from source for the first time is ... time consuming
- Build process is a wiki page for every distro – no usable build script

Prerequisite

- A newly installed RHEL/Rocky linux 8 x86_64 or aarch64 machine connected to the internet.
- **NOTE** It is suggested that you have at least 1GB of memory on the machine you are using for the build.
- **NOTE** Verify that SELinux is disabled.
- This manual uses the Rocky linux 8 distribution available at [x86_64](#) or [aarch64](#).
- It is assumed that Rocky linux 8 was installed as is, and nothing else was installed on it.

Overview

This document walks through the steps of patching the kernel, building Lustre and running a basic test of the complete system.

If you prefer to skip the kernel build steps and get right to building Lustre, you can download and install pre-patched server kernel RPMs from a recent [lustre-mas](#) least the `kernel`, `kernel-devel`, `kernel-modules` RPMs to install.

Process

Provision machine and installing dependencies.

Once RHEL/Rocky 8.10 is newly installed on an x86_64 or aarch64 machine login as user root.

1. Install the kernel development tools:

```
yum -y groupinstall "Development Tools"
```

2. Install additional dependencies:

```
yum config-manager --set-enabled powertools
dnf install -y gcc autoconf libtool which make patch diffutils file binutils-devel python38 python3-devel elfutils-devel 1:
yum -y install audit-libs-devel binutils-devel elfutils-devel kabi-dw ncurses-devel newt-devel numactl-devel openssl-devel
dnf install -y epel-release
dnf install -y ccache pdsh
dnf --enablerepo=ha install resource-agents
```

3. Install tools for kernel RPM build:

```
dnf install -y bpftool dwarves java-devel libbabeltrace-devel libbpf-devel libmnl-devel net-tools rsync
# May only be needed on RHEL9 derivatives:
dnf install -y python3-devel
```

4. Install e2fsprogs packages:

```
git clone "https://review.whamcloud.com/tools/e2fsprogs" e2fsprogs && cd e2fsprogs && git checkout v1.47.3-wc2 && ./configu
sudo make install
cd ..
```

Or download and install the following e2fsprogs packages from <https://build.whamcloud.com/>

Preparing the Lustre source.

Lustre VM Prep Process

- Create a VM with the distro you want to test...
- Get Lustre source...
- Find the wiki page for your distro...
- Install build packages (list may be outdated)...
- Download kernel source... (Is the wiki current on which version...? Probably not...)
- Install kernel config... (Is the kernel config current with the kernel you downloaded?)
- Patch kernel... Build kernel...
- Fix issues with kernel build...
- Build Lustre... (ldiskfs patches don't match? Might have to go back to "Download kernel source")
- Fix issues with Lustre build... (missing packages, etc)
- Lustre built!

Lustre VM Prep Process

- Install kernel...
- Reboot...
- Install Lustre...
- Build or install e2fsprogs... (Oh, there's a dependency chain issue with the distro e2fsprogs...?)
- Figure out what llmount.sh is...
- Run llmount.sh...
- Fix issues with VM... Is your network config right? Do you have all the necessary packages?
- Run llmount.sh again
- Success! You have a Lustre dev VM. If you need another distro or the kernel changes, just repeat the process.

Containerized Builds & VMs

- What if we built the kernel and Lustre in containers?
 - We can build for every distro, every arch, from anywhere.
- What if we had standardized image recipes?
 - Easily generate a VM image with the necessary packages installed and configuration in place
- What if we did Lustre dev in mini-vms, created inside a regular VM?
- What if we pre-published build artifacts?
 - Container build → Container download
 - Kernel build → Kernel download
 - Image build → Image download
 - But also publish build recipes, so you can build.

ltvm: Containerized Builds & Ephemeral VMs

- On a VM with nested virtualization support:
- `ltvm install`
 - Install dependencies, configure qemu and networking...
- `ltvm target fetch rocky9`
 - Download build container + kernel + image w/Lustre from github releases; ~1.5 GiB
- `ltvm create test-vm1`
 - Create default VM (rocky9)
- `ltvm llmount test-vm1`
 - Mount with built in Lustre
- **~3 minutes from “install” to “VM running Lustre”**

ltvm: Internals

- Lustre source + Kernel source + Container recipe + Image recipe
- Containers
 - Container recipe for every distro + arch combo (x86 + Rocky 9, ARM 64 + Rocky 10, etc)
 - Build container (docker/podman, etc)
 - Build the kernel in the container
 - Build Lustre in the container
- Create base image
 - Dockerfile for base image
 - Additional overlay for variants (eg, mofed)
- Base Image + Kernel + Lustre → ext4 image (bootable by qemu)

ltvm: Lustre and Network Config

- Easy to adjust VM configs:
 - `ltvm create pafvm1 --ost-disks 4 --mdt-disks 2 --vcpus 4 --mem 4096`
 - Uses default target – rocky9
 - Default is 1 TCP nic
 - `--nic` option supports `tcp`, `softroce`, or `passthrough`
 - `ltvm create pafvm1 --nic softroce --nic tcp`
 - 2 NICs in LNet (`tcp0(eth1)`, `o2ib0(rxe0)`); mgmt `eth0` is SSH-only
- Automatically wired into test framework
 - `mount` runs `llmount.sh`
 - OSTs and MDTs configured automatically
- Passwordless ssh by default:
 - `ssh myvm`

ltvm: Targets

- Can support every kernel and OS Lustre supports
 - Rocky 8, 9, 10 available as artifacts
 - Further variants can be built locally in one command:
 - `ltvm build all rocky9 --kernel 5.14-rhel9.6`

```
> ltvm target list
rocky8 (aarch64) -- Lustre server (ldiskfs backend), rocky 8.10
  Kernel          Local  Remote  State
  4.18-rhel8.10 (default)  -      ✓      fetch

rocky9 (aarch64) -- Lustre server (ldiskfs backend), rocky 9.7
  Kernel          Local  Remote  State
  5.14-rhel9.8      -      ✓      fetch
  5.14-rhel9.7 (default) ✓      ✓      ready
  5.14-rhel9.5      -      ✓      fetch

rocky10 (aarch64) -- Lustre server (ldiskfs backend), rocky 10.1
  Kernel          Local  Remote  State
  6.12-rhel10.1     -      ✓      fetch
  6.12-rhel10.0 (default) -      ✓      fetch
```

ltvm: Community development

- ltvm is new – first commits April 2026
- Lustre test configuration is complicated – many corner cases
 - Who other than Oleg can say their dev VMs can run all of the test-framework?
- Normally everyone does this for themselves...
- **Let's share the load.**
- ltvm is open for contributions – check out the code, try something, find it doesn't work
→ Make a PR on github.
 - Add new OS and kernel targets, fix tests, add features...
- Not owned by any company – Given freely to the community. (BSD licensed)
- Patrick and Marc push prebuilt artifacts for x86_64 and aarch64 for Apple Silicon Macs

ltvm: Updates since April (LUG 26) and Github

- 180+ additional commits and 12 new prebuilt artifacts
- The highlights among *many* little things:
 - Overall code hardening
 - New kernels rhel9.8 and rhel10.2
 - Deeper integration with AI agents: “Skills” are now installed along with `ltvm install`
 - QoL: Tab completion, vm resizing, cleanup for stale artifacts, and more
- Requirements:
 - Modern Linux VM (host agnostic) with **nested virtualization support**
 - Nested virtualization is pending for Mac (M3+ only) – for now run it natively on the Mac directly
- Check out, run `ltvm install`, `ltvm --help`, read README.md ...
or just have your agent use it

LLM Code and Review Tools

- With LTVM, agents can spin up mini VMs to complete various tasks
- LLM tools help agents to further interact with our services and more
 - Gerrit CLI: get patch state, comments, reviews, and interact with them
 - Jira CLI: get, search, comment, get attachments
 - Jenkins: build status, console output, retrigger, abort
 - Janitor and Maloo: get test results, get failures and its logs, find related bugs
 - drgn-tools: drgn-based Lustre vmcore analysis tools
 - Gerrit graph: Generate an interactive graph to visualize patch series including comments and state
 - lreview: Gerrit's AI review run locally
- LLM tools are made for LLMs (output JSON)
- LLM tools and LTVM allow a full workflow for LLMs:
 - Look at Jira ticket, spin up mini vm, reproduce (and possibly propose a fix)

Some examples

```
```bash
Simple lookup
jira --pretty get LU-10911

With comments – what humans said about the bug
jira --pretty get LU-10911 --comments 5

Just the comments
jira --pretty comments LU-10911 --limit 5
```
```

```
# Drilling into a specific session:
# overview: which suites passed/failed
maloo session <session-UUID> | jq
# only the failed suites + subtest errors
maloo failures <session-UUID> | jq
```

```
```bash
unresolved comments with code context – what the LLM reads to fix review feedback
gc comments https://review.whamcloud.com/61965 | jq

Full diff of the latest patchset (can be large – head it for the demo)
gc review https://review.whamcloud.com/61965 | jq | head -80
```
```

Gerrit Graph Visualizer

- Gerrit patch series are notoriously difficult and time-consuming to manage
- When patches are updated and the series runs out of sync, ordering is random
- To find the current state of patch, you must navigate to each individually
- The Gerrit graph visualizer aims to solve this:
 - It visually presents a patch series based on an LU ticket or Gerrit patch
 - See what depends on what and when
 - Node colors help to quickly understand if Jenkins, Maloo, Janitor failed, or a reviewer vetoed
 - Unresolved comments are shown
 - Can navigate to Gerrit, Jenkins/Maloo results, checkout, and cherry pick from the visualizer
 - Include related patches via hashtag and topics
- Processes large graphs (EC, wbc) in about a minute (fetching comments is opt-in)

LLM Tools: Updates since April (LUG 26)

- 230+ additional commits – the highlights:
 - lreview: Run Gerrit AI review locally with your LLM and with memory
 - 7 Agent Skills for Claude about Gerrit patch workflows, lreview, CI and crash triage, debug logs
 - Gerrit graph improvements including statistics for your patch series
 - Gerrit dashboard: An alternative dashboard integrated with our patch workflow
 - CLI improvements
 - Various fixes and hardening

LLM Tools: Ireview

- Ireview offers local AI review using the same prompts as Gerrit's AI review
- Integrated and tested with Claude and Codex
- Can run in parallel on several patches with Git worktrees (no need to push to Gerrit!)
 - A word of caution: This can exhaust your token quota **very** quickly
- By default, Ireview runs from scratch – no knowledge of previous runs
- Several options for controlling token usage
 - `--mode light` focuses on functional defects only
 - `--effort` controls reasoning effort for Claude and Codex
 - `-m/--memory` creates and uses a memory file documenting patch details and previous findings
- Ireview chat – Start an interactive session over an existing review
- Claude-integrated via a Skill

LLM tools: Running lreview

```
> lreview run --repo ../lustre-release -m 65921
65921 ps21 LU-19548 lfs: mirror extend support for EC

Reviewing 1 change(s), 5 in parallel, timeout 7200s each
prompts: /home/builder/workdir/remote-dev/llm_code_and_review_tools/review-prompts/kernel
         (via bundled submodule (/home/builder/workdir/remote-dev/llm_code_and_review_tools/review-prompts))
results: /home/builder/workdir/remote-dev/llm_code_and_review_tools/lreview-results
memory db: /home/builder/workdir/remote-dev/llm_code_and_review_tools/lreview-db
worktrees: /home/builder/workdir/remote-dev/ai_worktrees/lreview

reaped 1 worktree(s) left by killed runs
[65921_ps21] review started: LU-19548 lfs: mirror extend support for EC
running: 65921_ps21 27m51s 22.3M tok | done 0/1
[65921_ps21] 2 finding(s), severity low (31m05s, 23.9M tok, $18.99) -> gerrit-review-65921_ps21.json

=== Batch summary ===
65921_ps21 findings (2 finding(s), severity low) [23.9M tok, $18.99]
total: 23.9M tokens, $18.99

Human-readable reports:
/home/builder/workdir/remote-dev/llm_code_and_review_tools/lreview-results/markdown/65921_LU-19548_lfs_mirror_extend_support_for_EC_ps21.md

Review memory (what the patch does, findings, eliminated false positives):
/home/builder/workdir/remote-dev/llm_code_and_review_tools/lreview-db/65921-LU-19548_lfs_mirror_extend_support_for_EC.md (iteration 1)
```

LLM tools: lreview – Human Readable Report

```
# 65921 ps21 – LU-19548 lfs: mirror extend support for EC

- **Change:** https://review.whamcloud.com/c/fs/lustre-release/+/65921 (patchset 21, `3eb3f5ce67a1`)
- **Review:** 2 finding(s), severity **low**
- **Run:** opus, 23.9M tokens, $18.99, 31m05s
- **Review memory:** `/home/builder/workdir/remote-dev/llm_code_and_review_tools/lreview-db/65921-LU-19548_lfs_mirror_extend_support_for_EC.md`

## Overall assessment

Looks good. I could not find a correctness problem in the three new extend modes, the parity-only merge path, or the allocator change. Two optional points inline for whenever the patch is next refreshed; neither is a reason to re-spin on its own.

## Findings

### 1. `/COMMIT_MSG` (line 36)

(minor) lod_check_and_reserve_ost() is only reached from lod_ost_alloc_rr(), but test_8j sets lod.*.qos_threshold_rr=0, which keeps LQ_SAME_SPACE clear in ltd_qos_penalties_calc(), so lod_ost_alloc_qos() does the allocation itself instead of returning -EAGAIN.

The QoS path has its own lod_ec_check_data_usable() call that this patch doesn't touch, so neither the new hard constraint nor the new parity relaxation in lod_check_and_reserve_ost() is executed by test_8j.

If the patch is refreshed, should this paragraph name the QoS path instead, and would a second pass with qos_threshold_rr=100 be worth adding so the changed round-robin code is covered too?

### 2. `lustre/utils/lfs.c` (line 2570)

(minor) The target component's pattern is inspected for DoM here, but not for overstripping.

lod_comp_prep_create() carries LOV_PATTERN_OVERSTRIPING from a data component onto its parity only in the !is_parity branch, i.e. while the data component itself is being created. In the extend flow the data component is already instantiated, so the post-merge resync preps only the parity and that propagation never runs: a parity attached to a '-C' data mirror stays plain, unlike the one 'lfs setstripe -C --ec' ends up with.

I couldn't turn that into a concrete failure – lod_ec_add_parities() covers the short-OST case either way – so this is a question rather than a defect: should a '-C' target be refused here like DoM is, or should the overstripping be set on the parity component when it is built?
```

LTVM and LLM tools demo

Closing

- LTVM and LLM tools are open source (BSD-licensed) and are continuously improved
- https://github.com/lustre-tools/llm_code_and_review_tools/
 - LLM tools and Gerrit graph visualizer (part of Gerrit CLI)
- <https://github.com/lustre-tools/lustre-test-vms>
- <https://github.com/lustre-tools/portal>
 - Brand new! A web portal for tracking patch-series (Gerrit graph), the Dashboard, and Promises (WIP)
 - Will include further useful tools in the future: Marc's portal hosted at <https://lustre.lefthy.de>
- Made for and by the community
- All projects are open for contributions
- Bring your LLM! If you see issues, make a PR on Github
- [Join us on Slack!](#) (part of the Linux Lustre Client Slack)

Thank You

Marc Vef | mvef@ddn.com

Patrick Farrell | pfarrell@thelustrecollective.com

