Read-Mostly: Letting AI Assistants Run Real Slurm Jobs Safely

A Field Note on ursa-bifrost and ClusterWand

Charles E. Forsyth III
Research Computing
University of California, Riverside
Riverside, California, USA
charles.forsyth@ucr.edu

Abstract

Between 1 October 2026, when the specification and first release were written, and 27 October I built ursa-bifrost, a Model Context Protocol (MCP) server and command-line tool (CLI) that lets AI assistants work on Ursa Major, UC Riverside Research Computing's Slurm cluster on Google Cloud. It reached v0.10.1 in 42 tagged releases. Assistants can read what a researcher could read about their own jobs. They can change state only through a two-step submit that is capped, bound to the exact plan, and logged. By design there is no shell tool.

The main lesson is about testing. A mutation script switches off each of 219 safety guards in turn and requires some test to fail, so each guard is known to be needed. An outside review found three approval checks that existed, passed every test and were wrong. Live tests found a worse bug: a confirm token five processes could redeem at once. Real use so far is one person, me, plus programs running as me. On 28 October I forked the code as ClusterWand, a public tool that runs over a person's own SSH login.

CCS Concepts

• **Security and privacy** → **Software and application security**; • **Software and its engineering** → *Software testing and debugging*; • **Computing methodologies** → *Distributed computing methodologies.*

Keywords

Model Context Protocol, Slurm, AI assistants, prompt injection, mutation testing, research computing

Author note

The work was directed by me and carried out with AI agents under written briefs, independent checks and explicit approval steps. Figures come from the two repositories' git history, tags, changelogs and specifications, the mutation script, my audit log, the deep-research specification, the cluster change log and two review records. This note was drafted with AI assistance from those records and reviewed by me. Example job numbers are representative values, not records of specific researchers' work.

Cite this note. C. E. Forsyth III. 2026. Read-Mostly: Letting AI Assistants Run Real Slurm Jobs Safely. Field Notes.
doi:10.5281/zenodo.23752286

1 Context

Most of what an AI assistant could usefully do on a cluster is read: why job 236 failed, which partition is idle, whether a batch script asks for more memory than the node has, which module has LAMMPS (a molecular dynamics code) with graphics processor (GPU) support. Those are the bulk of "my job failed" support tickets, and the digging behind each one is the same every time: the accounting record, the end of the log, the partition state. The ursa-bifrost specification started from that, and from two bugs in the deep-research engine's own Secure Shell (SSH) layer, one of which kept an idle node running for nobody.

The easy way is to give the assistant an SSH session. I ruled that out in the specification before any code was written. With a shell I can no longer say which commands run, with what arguments, at what cost, or what record is left. And a job log is text written by programs and people, so someone can plant a line in it aimed at the model that reads it [1].

Who uses it so far: very few. Ursa Major is a young cluster. Its Tier 1 partitions (Arm and Tensor Processing Unit (TPU) nodes for projects approved after a consultation) and its regular partitions have few users, and the first allocation, on 27 October, was for Research Computing's own pilot lab. A 27 October survey of lab cloud projects found that only one person had used the cluster through bifrost in the previous 30 days, and that was me. The deep-research Lab, a program that plans and runs experiments on the cluster, is designed to be the main caller, but it runs under my identity, so the log cannot separate it from me. As of 28 October 2026, my laptop's audit log for 27 and 28 October held 152 calls, all mine, through the CLI: 97 job lookups, 24 plans and 10 confirmed submits. I would rather get the safety design right while the user list is this short.

2 Related Work

Assistants, tools and injection. MCP [2] standardizes how an assistant discovers and calls tools over JSON-RPC; its tools section says there should always be a human in the loop able to deny a tool call, but leaves enforcement to each client. Greshake et al. showed that text an application retrieves can carry instructions that steer the model reading it [1], and the OWASP list for LLM applications names both prompt injection and "excessive agency" as top risks [3]. Bifrost takes the second risk as the one it can control: it cannot stop a model from being misled, so it limits what a misled model can do. HPC-GPT fine-tunes a language model for HPC questions and code [4], and Li sets out the threat of an agent acting under a user's own HPC credentials being steered by text in job logs, proposing a benchmark rather than a tool [5]; bifrost is one deployed answer to that threat.

Protection principles. The design leans on old ideas. Saltzer and Schroeder's least privilege and complete mediation [6] map to a

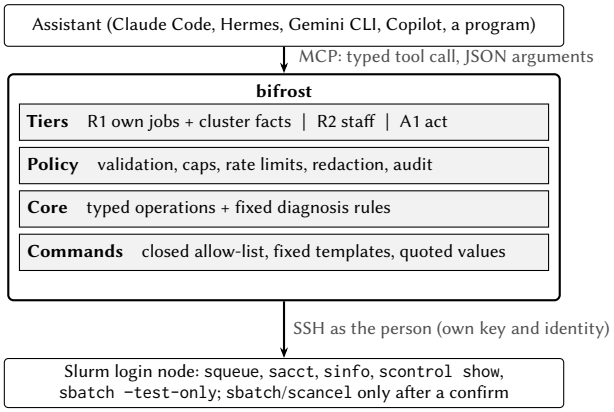


Figure 1: The layers of ursa-bifrost. Every request passes the tier check and policy before the core builds a command from the allow-list. A model can supply only validated values; it never supplies command text.

closed command list and a check on every call. Hardy's confused deputy [7] describes an agent misusing authority it holds for someone else, which is the risk when an assistant acting for a person reads text written by others. The tiers are a small role-based scheme in the sense of Sandhu et al. [8].

Cluster access without a shell. Slurm [9] offers a REST daemon, `slurmrestd`, which authenticates with JSON Web Tokens [10–12], and its JWT guide describes authenticating proxies placed in front of it [11]. Slurm's own association and quality-of-service limits apply to every submission path [13]. Open OnDemand gives people web access to clusters with no command line [14]. For job efficiency, Open XDMoD [15] and Princeton's `jobstats` [16] report usage after the fact for people and administrators.

Testing the tests. Mutation testing, proposed by DeMillo, Lipton and Sayward [17] and surveyed by Jia and Harman [18], judges a test suite by whether it detects deliberately injected faults. Bifrost applies it narrowly: one hand-written mutation per safety guard rather than generated mutants across the code.

Identity and data movement. SSH [19] carries a single command string in its `exec` request, which shapes the quoting rules below. Google Cloud's Identity-Aware Proxy forwards SSH through an authenticated tunnel [20], OS Login ties SSH access to each person's cloud identity [21], and Cloud Storage signed URLs grant time-limited access to one object [22].

3 The Design: Read Freely, Act Twice

Bifrost is one Go binary with three faces: an MCP server over standard input and output for a laptop, the same server over HTTP with Google sign-in for a hosted copy on Cloud Run, and a CLI where every tool is also a command with `-json`. It holds no language model. It runs scheduler queries and fixed diagnosis rules, and the calling assistant does the reasoning. That keeps it cheap and testable. Figure 1 shows the layers.

Rule one: a closed list of commands. A command can only be made by a constructor in one file, and a test, `TestAllowListIsClosed`, pins the set of remote programs to 17 names. `scontrol` may only show, hold or release. A bare `sbatch` must be `-test-only`. Arguments are validated first (a job id must be digits). SSH has no argument vector: it carries one command string to the remote login shell [19]. So the remote line is a fixed template, and each value is either checked against a strict pattern or single-quoted by one function, `Quote`. Some templates pass values as positional parameters to a fixed `bash -c` string. Safety rests on validation and quoting. Anyone who adds a command has to edit the test on purpose.

Rule two: tiers decide which tools exist. R1 is any signed-in person: their own jobs and public cluster facts. R2 is staff: every user's jobs, usage and waste, node health, and draft ticket replies that are never sent. A1 is opt-in per person: submit, cancel, hold and release their own jobs. A caller does not see a tool for a tier they lack, so a read-only assistant cannot even try to submit. On 2 October the hosted server listed 36 tools with all tiers: 19 R1, 7 R2 and 10 A1. Admin work, such as node states, reservations and accounts, is not exposed.

Rule three: acting takes two calls. `job_submit` does four things:

- runs the static script check;
- enforces caps on nodes per job, hours per job, worst-case dollars per job and per day, and submits per day;
- asks Slurm for a `-test-only` estimate;
- returns a plan with a single-use `confirm_token`, bound to a hash of the exact plan and valid for 10 minutes.

Nothing is submitted. Only `job_submit_confirm` with that token submits, after it re-checks the hash and the caps. So the bytes confirmed are the bytes planned. The token does not prove that a person read the plan, because the assistant decides how to summarize it. The specification also counts on the client's own approval prompt, but bifrost cannot enforce that prompt and some clients approve tools automatically, so I count only the token and the caps as guarantees. The example config ships caps of 4 nodes, 24 h, $25 per job, $50 per day and 20 submits per day; a site sets its own.

Every result carries the commands that produced it, so an answer can be checked against the cluster.

`job_explain` is the main diagnosis tool. It runs 11 state rules over the accounting record (out of memory, timeout, node failure, a signal exit and so on) and about 20 log rules over the redacted log tail (a missing Python package, a module that does not exist, a binary built for a newer processor, a full disk). Each finding carries the evidence that fired it and a suggested fix. The rules were seeded from the failure classes the deep-research engine had already met, and each log rule has a sample in a test.

4 Threats, and What Stops Them

Job names, log lines and script bodies come back inside a field named `untrusted`, and tool descriptions tell the client that field is data. That label is a hint to the model. It does not stop prompt injection [1]. Containment comes from capability: no tool acts on content it read, writes need a token, and caps apply to every write. Table 1 maps each threat I planned for to the control that stops it, and to what is left.

Table 1: Threats to the AI path and the controls in bifrost.

Threat	Control	What is left
Model error: wrong job, wrong script	Two-step token bound to the plan hash; caps	A capped job that is still the wrong job
Injected text in a log or job name	No tool acts on what it read; read-only tiers hide write tools	The model may propose a harmful plan that a tired person approves
Injection across tools in one session	None in bifrost	A staff assistant that reads another user's log and also has mail or chat tools loaded could be steered into those tools
Buggy or auto-approving client	Token and server caps; the client prompt is not relied on	Submits up to the caps
Program that confirms its own jobs	Program client with its own ceiling, budget and day cap	Spend up to that ceiling
Person bypassing bifrost	None; this is the cluster's job	Normal `sbatch` over their own login
Compromised hosted server	Sessions, login keys and approval state sealed with AES-256-GCM; pinned host keys	Each user's sealed Google refresh token carries that user's cloud rights

Two of those rows need a word. The hosted server asks each user for Google's broad `cloud-platform` scope, because the Identity-Aware Proxy (IAP) tunnel [20] and OS Login [21] key calls need it; the repository's security notes record this as known and accepted, and admins can use the CLI instead. And staff-tier tools send other users' job metadata to whatever model the staff member's assistant uses. Whether staff tools may read other users' logs at all is still an open question in the specification, as is how long the audit log is kept and who reviews it. I have not run an injection test against a staff assistant with other tools loaded.

Other approaches exist, and I chose against them for stated reasons. Open OnDemand [14] already gives web access to Slurm without a shell, and runs on Ursa Major; the specification puts a job portal out of scope for that reason, and an MCP server serves a different client, the assistant. The Slurm REST daemon (`slurmrestd`) with per-user JSON Web Tokens (JWT) [10, 11] was the planned backend; I dropped it once IAP and OS Login gave each person their own identity over SSH, which needed no new service. Slurm quality-of-service (QOS) limits [13] would enforce caps for every path, not just this one; Ursa Major has none beyond `normal` today, and long term the caps belong there. For diagnosis, `seff`, Princeton's `jobstats` [16] and Open XDMoD [15] (which Ursa Major runs) report job efficiency; `job_explain` adds log rules and a fix aimed at an assistant. The specification lists several open-source Slurm MCP servers on GitHub; when I looked, none advertised tiers, approval or audit, and I have not evaluated them further.

5 Guards, and Proving They Work

A green test suite does not tell me whether any test depends on a given guard. If I delete the owner check and every test still passes, the owner check is unguarded, whatever the coverage report says. So bifrost has `scripts/mutation_check.sh`, a narrow form of mutation testing [17, 18]. Each line names a guard, the file it lives in, the exact code, and a replacement that disables it: the tier check becomes `if true`, the token's single-use delete becomes a no-op, redaction stops replacing anything. The script applies one mutation, runs the full test suite, and expects it to fail. A mutation the tests survive fails the run. A pattern that no longer matches the code is reported as BROKEN and also fails the run. It runs by hand, not in continuous integration. Table 2 shows the guard count by release. Each new safety check shipped with a guard in the same release; v0.8.0 jumped because shared-partition support added share pricing and core rules across many files, with 12 releases between it and v0.3.1.

Table 2: Mutation guard counts by release.

Release	Date	Guards	What was added
v0.3.1	1 Oct	26	Two-step submit, caps, live test fixes
v0.8.0	2 Oct	113	Shared partitions: core rules, share pricing
v0.9.0	2 Oct	128	Program clients with tier ceilings
v0.9.14	5 Oct	208	Fast accounting reads; older gaps closed
v0.10.1	27 Oct	219	Allocation framing (warn-only)
ClusterWand v0.1.0, 28 Oct		148	Fork: 71 guards removed with the hosted code

On 5 October the full run found one guard that had been BROKEN since v0.9.6 (a function signature changed, so the pattern no longer applied) and five guards no test killed. One of those mattered: the background renewal of a person's cluster login key wrote the renewed key back to its store, and no test noticed if it did not, because the in-memory store used in tests hands back the same object it was given. Each of the five got a test.

Mutation testing only covers what I thought to guard. Two other kinds of check found what I had not thought of.

Live campaigns. On 1 October, after the act tier shipped, I ran 13 real jobs, 16 negative CLI cases and 22 MCP checks against the cluster. The worst find: five parallel `bifrost confirm` calls with one token all reached the cluster. The lock was per process, and only a folder-exists check stopped four of the five jobs. The fix is an exclusive file lock around every read and write of the approval state, with a race test that runs the built binary. The same campaign found a rule-ordering bug in redaction, fixed by changing which rule runs first. A day later, a 30-call parallel burst failed 9 calls: the

login node allows 10 sessions per connection, and bifrost now uses at most 8 per person.

An adversarial review. On 20 October five models each reviewed bifrost v0.9.15 and the cluster configuration through five lenses (security, code, operations, cost governance, and drift between spec and code): 25 reviews, 374 findings, 337 after clustering. I checked by hand every finding I acted on. 17 were confirmed and ranked, and 5 "critical" claims were checked and rejected; one model said the audit log wrote signed URLs in clear, but every command line passes through the redactor first. The report does not classify the remainder one by one. About 5% of distinct findings held up. v0.9.16 fixed the confirmed bifrost findings the same day. Three were gaps in the approval path:

- A plan with a partition override dropped every check error that mentioned the word "partition", so a 20-core script moved onto a 16-core partition planned cleanly. The override now gets the full static check.
- The submits-per-day cap was checked when a plan was prepared, not when it was confirmed. Preparing several plans and then confirming them all went past the count. The dollar cap had held.
- A time limit that is not a Slurm time, such as `1hour`, parsed as 0 minutes. That hid the "no time limit" warning and priced the job at $0. It is now refused.

The mutation guards missed all three, because each check existed but was written wrong.

6 Files Without a Shell

Section 18 of the specification covers files, and the design choices are mine from 1 October. Inputs are staged: `upload_prepare` returns a signed upload link [22], valid for 15 minutes, for one object in a private bucket, signed for a maximum size so the storage service itself rejects a larger file. The bytes go straight from the person to storage and never pass through bifrost, the model or the login node. When the job is submitted with that upload id, the plan hash covers the object's generation and size, so the confirm token approves exactly those bytes. On confirm, bifrost inserts a fixed block after the `#SBATCH` lines that fetches each file with a read-only link into `inputs/`, and fails the job with exit code 66 if a fetch fails. Compute nodes hold no credentials, and the links expire. Outputs go the other way through signed download links that last 60 minutes, and the bucket deletes everything after 7 days.

The read-only reasons people reach for a shell got their own tools: `storage_usage`, `files_list` and `files_read` under the person's own home and scratch, refusing hidden and credential-like files; and `env_check`, which loads modules inside a one-core job on a small always-on partition, never on the login node. `interactive_help` writes the exact command for a real interactive session, because people who need a shell already have one under their own identity. Logs come back in windows of up to 1,000 lines with line numbers and a `grep`.

Paging also forced a useful measurement. A routine status check on 5 October took about 145 s of bifrost time, and `sacct -json` for one person over 7 days took 19.6 s. Plain text for every user over the same 7 days took 0.2 s. A field-by-field comparison of 745 live jobs found no difference that matters, and list reads moved to plain text in v0.9.14. That change also makes the fork possible, since older Slurm versions have plain text but not the same JSON.

7 When a Program Confirms Its Own Jobs

The two-step rule assumes a person reads the plan. The deep-research Lab breaks that assumption on purpose. Its workflow is plan, pilot, full run, then judgment against criteria written down in advance, and asking a person to approve each step would defeat it. I covered the Lab's move from SSH to bifrost in the field note *One Day on the Research Stack* [23]; here I only want the safety argument.

Pressing Submit on a reviewed plan in the Lab counts as the person's approval. The Lab then calls `job_submit` and confirms the token itself only when every guard holds:

- **Lineage.** The job comes from the Lab's own Submit path or the one automatic partition switch of a run submitted that way. Re-plans, "Fix with AI" and reruns go back to draft.
- **Same script.** Bifrost's hash of the script must equal the hash of the script the Lab just built. Otherwise the token is never used.
- **Per-run budget.** Bifrost's worst case must be at most three times the reviewed estimate (with a $1 floor) and never over a configured per-run limit, $10 by default.
- **Per-run count.** At most 6 bifrost submissions per run.
- **Server caps on top.** The Lab signs in as a pre-registered program client with its own tier ceiling, call budget, day cap and spend ledger, separate from the person's.

The client-side guards protect against the Lab's own bugs. The server-side caps protect against everything else, including those guards being wrong. A program's tools are the person's tiers intersected with the program's ceiling, so a ceiling can take tiers away and never add one. My desktop operator app, Ultra, signs in the same way with read tiers only, even though I hold A1.

8 ClusterWand: the Public Fork

Bifrost is public, but it is shaped like Ursa Major: Google sign-in, an IAP tunnel with OS Login keys, Cloud Storage staging, partition names in code, and cloud list prices as the cost model. A plan written on 28 October counted the Ursa assumptions in the v0.10.1 code; partition names alone were hard-coded in 22 places. Making those pluggable inside bifrost would have put the live service at risk with every step, and I had said not to touch it. So I forked with full history, so that blame and the guards come along, and fixes are cherry-picked by hand between the two.

ClusterWand was created on 28 October from bifrost v0.10.1, and v0.1.0 was released the same day. It is SSH only. It runs on the person's own machine and reaches the login node through their own OpenSSH config, keys and agent, over one multiplexed connection. Everything hosted is gone: the HTTP server, Google sign-in, the IAP backend, Cloud Storage staging and the dashboard, with their tests and 71 of their guards. That leaves 148. The first full mutation run on the fork had 4 survivors, because their only tests had lived in the removed packages; after 4 new tests the rerun killed all 148.

Its README is plain about what v0.1.0 is not yet. The tools still expect a cluster catalog file describing partitions, prices, modules

and recipes, and the defaults are Ursa Major's. On another cluster a person writes that catalog by hand in their own home folder; nothing is installed on the cluster. Both tools are designed to be adapted to other Slurm clusters, but the only cluster either has run against is Ursa Major.

9 What Went Wrong

- **The confirm token was not single-use across processes.** Five parallel confirms reached the cluster. Only an unrelated folder-exists check stopped four of the jobs. A file lock and a race test fixed it, but the original design had been reviewed and looked right.
- **Checks written wrongly.** The partition-override filter, the confirm-time submit count and the 0-minute time limit all passed tests and mutation guards for about 18 days, from v0.8.0 on 2 October to the review on 20 October.
- **The caps only govern bifrost.** Anyone with a normal login can still run `sbatch` directly. The review also raised cluster configuration items. Most are fixed: Spot nodes are now deleted on preemption, the Arm partition is whole-node as decided, the rebuild is scripted, the node service account lost its broad Editor role on 21 October, and partitions got a default time of 24 h and a maximum time (now 30 days on most). Job privacy settings and backups were on the follow-up list, tracked in the cluster change log separately from bifrost. Bifrost's "own jobs" limit applies only to its own tools.
- **Cost guards that understate.** On a shared partition the worst case is the share of the node a job holds. That is the right display, but one small job can keep a whole node awake, so the cap understates what the cloud bills. This is recorded and not yet changed.
- **A mutation guard can rot.** One sat BROKEN for several releases before a full run caught it. The script now fails on BROKEN, but only when someone runs the full set.
- **The fork lost a safety limit it did not know it had.** The 8-session cap per person lived in the IAP backend. ClusterWand's plain SSH backend has no such cap yet, so a burst can hit the server's session limit. The fork's SSH guide says so; it is not fixed.
- **Little real use.** 42 releases in 27 days, and one human user. The evidence is test evidence and my own jobs, not researcher traffic.
- **Unverified elsewhere.** Neither tool has run on a second cluster, an older Slurm version, or a cluster that asks for a second factor on every login.

10 Lessons

(1) **Test races with the real binary.** The worst bug lived between processes, where unit tests could not see it.
(2) **Review each check for correctness, not only presence.** Mutation guards prove a check is needed; three wrong checks passed 200-odd of them.
(3) **Get outside reviewers, and check what they say.** They found checks written wrongly, and 5 claims labelled critical were false.
(4) **Put a server-side ceiling under every program that approves its own work.** Only the server catches the client's guards being wrong.
(5) **Do not give an assistant a shell.** The other guarantees depend on a closed set of commands.
(6) **Make acting take two calls bound to one plan.** A hash-bound, single-use token stops a model from planning one thing and submitting another.
(7) **Fork instead of generalizing a live service.** It keeps refactoring away from the production service.

11 What's Next

- ClusterWand: move every Ursa constant into a site profile, with tests that Ursa outputs stay byte-identical to bifrost v0.10.1; then text-format Slurm parsing for older versions, then a read-only `discover` that drafts the catalog.
- Run every read tool and one approved test job against a second cluster through a normal researcher login.
- A session cap on ClusterWand's plain SSH backend.
- Bifrost: price the cap check at the whole node on shared partitions, keeping share pricing for display.
- An injection test against a staff assistant with other tools loaded.
- The measurement that would settle the safety claim: researchers other than me using it for a term, counting the share of plans rejected or edited before confirm and any submit that went beyond what its plan showed.

Sources

Public: https://github.com/UCR-Research-Computing/ursa-bifrost (code, tags, changelog, security notes, `docs/SPEC.md` Draft 8, `scripts/mutation_check.sh`);
https://github.com/UCR-Research-Computing/clusterwand (v0.1.0 release, `docs/SSH.md`);
https://github.com/charles-forsyth/deep-research. Private records: the 20 October adversarial review report (25 reviews); my local bifrost audit log, 27–28 October; the 27 October survey of lab cloud projects; the 28 October ClusterWand plan (Draft 4) and F0 build log; the Ursa Major cluster repository, operations specification and change log.

References

[1] Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec '23)*. ACM, 79–90. doi:10.1145/3605764.3623985

[2] Model Context Protocol. 2025. Model Context Protocol Specification, Version 2025-06-18. https://modelcontextprotocol.io/specification/2025-06-18. Accessed 28 October 2026.

[3] OWASP Foundation. 2025. OWASP Top 10 for Large Language Model Applications. https://owasp.org/www-project-top-10-for-large-language-model-applications/. Accessed 28 October 2026.

[4] Xianzhong Ding, Le Chen, Murali Emani, Chunhua Liao, Pei-Hung Lin, Tristan Vanderbruggen, Zhen Xie, Alberto Cerpa, and Wan Du. 2023. HPC-GPT: Integrating Large Language Model for High-Performance Computing. In *Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis*. 951–960. doi:10.1145/3624062.3624172

[5] Jie Li. 2026. Trusted Credentials, Untrusted Behavior: Benchmarking LLM-Agent Security in High-Performance Computing. arXiv:2607.18485. https://arxiv.org/abs/2607.18485

[6] J. H. Saltzer and M. D. Schroeder. 1975. The Protection of Information in Computer Systems. *Proc. IEEE* 63, 9 (1975), 1278–1308. doi:10.1109/PROC.1975.9939

[7] Norm Hardy. 1988. The Confused Deputy. *ACM SIGOPS Operating Systems Review* 22, 4 (1988), 36–38. doi:10.1145/54289.871709

[8] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. 1996. Role-Based Access Control Models. *Computer* 29, 2 (1996), 38–47. doi:10.1109/2.485845

[9] Andy B. Yoo, Morris A. Jette, and Mark Grondona. 2003. SLURM: Simple Linux Utility for Resource Management. In *Job Scheduling Strategies for Parallel Processing (Lecture Notes in Computer Science, Vol. 2862)*. Springer, 44–60. doi:10.1007/10968987_3

[10] SchedMD. 2026. Slurm Workload Manager: REST API Details. https://slurm.schedmd.com/rest.html. Accessed 28 October 2026.

[11] SchedMD. 2026. Slurm Workload Manager: JSON Web Tokens (JWT) Authentication. https://slurm.schedmd.com/jwt.html. Accessed 28 October 2026.

[12] M. Jones, J. Bradley, and N. Sakimura. 2015. *JSON Web Token (JWT)*. RFC 7519. RFC Editor. doi:10.17487/RFC7519

[13] SchedMD. 2026. Slurm Workload Manager: Resource Limits. https://slurm.schedmd.com/resource_limits.html. Accessed 28 October 2026.

[14] Dave Hudak, Doug Johnson, Alan Chalker, Jeremy Nicklas, Eric Franz, Trey Dockendorf, and Brian McMichael. 2018. Open OnDemand: A Web-Based Client Portal for HPC Centers. *Journal of Open Source Software* 3, 25 (2018), 622. doi:10.21105/joss.00622

[15] Jeffrey T. Palmer, Steven M. Gallo, Thomas R. Furlani, Matthew D. Jones, Robert L. DeLeon, Joseph P. White, Nikolay Simakov, Abani K. Patra, Jeanette Sperhac, Thomas Yearke, Ryan Rathsam, Martins Innus, Cynthia D. Cornelius, James C. Browne, William L. Barth, and Richard T. Evans. 2015. Open XDMoD: A Tool for the Comprehensive Management of High-Performance Computing Resources. *Computing in Science & Engineering* 17, 4 (2015), 52–62. doi:10.1109/MCSE.2015.68

[16] Princeton University Research Computing. 2026. jobstats: A Slurm-Compatible Job Monitoring Platform. https://github.com/PrincetonUniversity/jobstats. Accessed 28 October 2026.

[17] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. 1978. Hints on Test Data Selection: Help for the Practicing Programmer. *Computer* 11, 4 (1978), 34–41. doi:10.1109/C-M.1978.218136

[18] Yue Jia and Mark Harman. 2011. An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions on Software Engineering* 37, 5 (2011), 649–678. doi:10.1109/TSE.2010.62

[19] T. Ylonen and C. Lonvick. 2006. *The Secure Shell (SSH) Connection Protocol*. RFC 4254. RFC Editor. doi:10.17487/RFC4254

[20] Google Cloud. 2026. Using IAP for TCP Forwarding. https://cloud.google.com/iap/docs/using-tcp-forwarding. Accessed 28 October 2026.

[21] Google Cloud. 2026. About OS Login. https://cloud.google.com/compute/docs/oslogin. Accessed 28 October 2026.

[22] Google Cloud. 2026. Signed URLs. https://cloud.google.com/storage/docs/access-control/signed-urls. Accessed 28 October 2026.

[23] Charles E. Forsyth III. 2026. One Day on the Research Stack. Field Note. https://charles-forsyth.github.io/papers/2026-10-03-one-day-on-the-research-stack.html