MACADMINS 2026

Security Called:
How I Turned a Scripting Disaster into a GitOps Pipeline

Matt Parker · Senior IT Engineer, PrizePicks

github.com/maparker/shikomi

“Hey, got a sec?”

S **Security Team** 4:47 PM

Hey Matt, got a sec? Our scanner flagged a credential pattern in one of the repos you committed to. Can you jump on a call?

What I'd done

I'd hardcoded an API token into a shell script while testing.

Told myself I'd clean it up before committing. I didn’t.

I knew better. That’s the part that stings.

Three days passed. The token sat in the repo.

Nothing exploded. Nothing alerted. Until a scanner ran.

I didn't catch it. **Security did. After the fact.**

The questions I couldn't answer

01 What version is running on the fleet right now?

02 When did I last change it?

03 Who reviewed it before it went out?

Nobody taught me to start here. There's no onboarding for this. You write the script, it works, it goes into production. You learn the rest after Security calls.

What Mac Admins ship is production code with a .sh extension

The scripts you write run:

- With root or admin privileges
- On every machine in the fleet
- During provisioning, updates, and inventory checks
- Often handling API tokens for Jamf Pro, Slack, Okta, or other services
- Without sandboxing, staged rollout, or rollback

> **A bad script is a fleet-wide incident. The blast radius is bigger than most application code.**

Three gaps the Slack call exposed

01 Secrets end up in code.

02 Shell is uniquely easy to get subtly wrong.

03 Nobody knows what version is actually running on the fleet.

It’s not a people problem

Git doesn’t ship with secret scanning enabled.
Your MDM vendor doesn’t tell you how to version your scripts.
You have to wire all of this up yourself.

> **Most people don’t. Until something goes wrong.**

段取り八分、仕事二分

Eight parts preparation, two parts execution.

Shikomi is named after 仕込み, the Japanese culinary term for *mise en place*. The preparation that happens before you start cooking.

None of this technology is new. It simply helps you prepare properly.

Shikomi doesn’t invent anything. It wires tools together so you don’t have to.

> **The guardrails are there before you write a single line of logic.**

Five decisions. Made once. Built in forever.

01 Script type

Regular script with full logging and parameters, or Extension Attribute for Jamf inventory reporting.

02 Jamf parameters ($4–$11)

Label each one. Mark it as a secret or not.

Non-secret
Gets a default local value for testing

Secret
No default. Choose storage: 1Password, ~/.jamf_secrets, or manual

03 Static macOS variables

12 pre-built system variables by number: serial, logged-in user, OS version, and more. Enter the numbers you need.

04 GitHub Actions

Two optional workflows: PR validation (ShellCheck + version checks on every pull request) and deploy to Jamf Pro on merge via the Jamf Pro API.

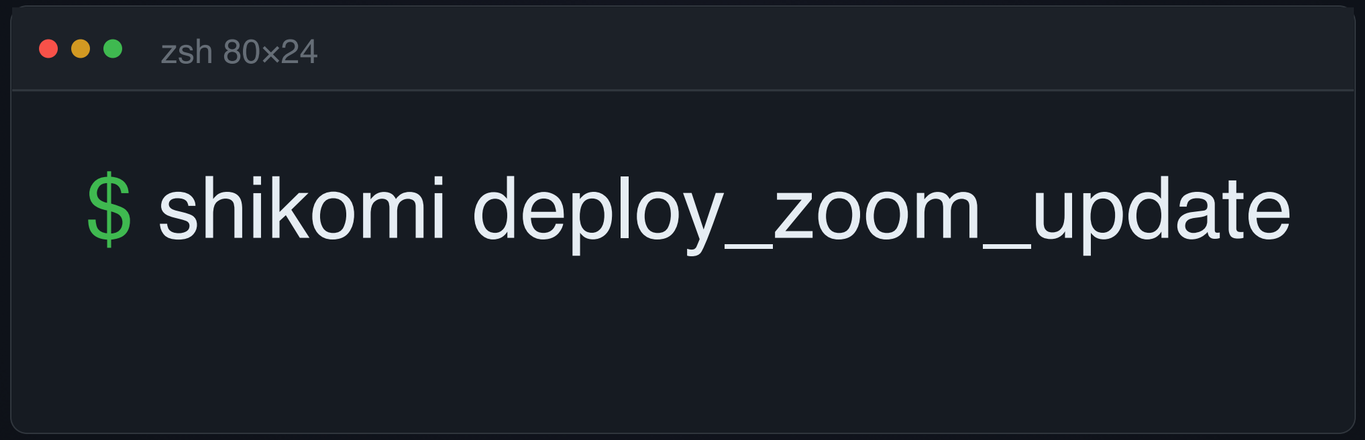
05 GitHub repo

Optional private repo creation via the GitHub CLI. Initialized, committed, and pushed before you leave the wizard.

The same script run locally for testing and deploys to production. No code changes between environments.

DEMO 1

Shikomi in action



```
> source ~/.zshrc
> c
```

Nine files. One commit. Before a single line of logic.

```
[master (root-commit) 8cad9b4] Initial commit: Scaffolding for
test-2
 9 files changed, 526 insertions(+)
 create mode 100644 .github/workflows/deploy-to-jamf.yml
 create mode 100644 .github/workflows/validate-version.yml
 create mode 100644 .gitignore
 create mode 100644 CHANGELOG.md
 create mode 100644 CLAUDE.md
 create mode 100644 README.md
 create mode 100644 SESSION_DIARY.md
 create mode 100755 test-2.sh
```

test-2.sh	Your script. Version header, structured logging, secrets wired up, parameters defined.
deploy-to-jamf.yml	Deploys your script to Jamf Pro via API on merge to main.
validate-version.yml	Runs ShellCheck and version checks on every PR.
.gitignore	macOS-aware. Secrets files excluded by default.
README.md CHANGELOG.md	Version-aligned with the script from the start.
CLAUDE.md SESSION_DIARY.md	Optional Claude Code configuration. Project conventions loaded before your first session.

526 lines of scaffolding you did not have to write. A project that was already production-ready before you opened the script.

Two gates. Neither optional.



Gate 1 catches you before you push. Gate 2 catches you if you bypass Gate 1. You need both.

DEMO 2

Gate 1 in action

01 A hardcoded secret gets added to the script.

02 Gitleaks catches it before it leaves the machine.

03 The fix wires it to 1Password instead.

```
> source ~/.zshrc
> c
```

Gate 1: the secret never leaves your machine

```
$git commit -m "Add Zoom auto-update logic"

Gitleaks.....................................Failed
- hook id: gitleaks
- exit code: 1

Finding: SLACK_TOKEN="xoxb-onwDRyHtYTaCNgJj9j7WgBsNJyD7UPFz393VeN"
Secret: xoxb-onwDRyHtYTaCNgJj9j7WgBsNJyD7UPFz393VeN
File: deploy_zoom_update.sh
Line: 31

[ERROR] Commit blocked. Remove the secret and commit again.
```

No commit. No push. No Slack message at 4:47 PM.

Gate 1: the bug stays on your machine

THE CODE

```
LOGGED_IN_USER=$(stat -f%Su /dev/console)
cd /Users/$LOGGED_IN_USER/Library/Logs
echo "Setup complete" >> setup.log
# ^ cd can fail silently -- SC2164
```

SHELLCHECK OUTPUT

```
shellcheck.....Failed

deploy_zoom_update.sh:45: warning [SC2164]

Use 'cd ... || exit' in case

cd fails.

Did you mean:

cd /Users/$LOGGED_IN_USER/Library/Logs || exit
```

> ShellCheck doesn't just say “bad.” It tells you the fix.

One version. Four places. Always in sync.

Mac Admins ship scripts inside packages. Version drift is a real problem.

- ! The script header says 1.2.0.
- ! The package is actually 1.1.4.
- ! Jamf shows something else entirely.

Which one is running on the fleet?

> **bump-version solves this. One command updates every version reference simultaneously.**

DEMO 3

bump-version in action

zsh 80×24

```
$bump-version patch "Fixed privilege check on non-admin accounts"
```

```
> source ~/.zshrc
> export PATH=/usr/local/bin:$PATH
> c
```

One command, every version in sync

```
$bump-version deploy_zoom_update.sh patch “Fixed privilege check on non-admin accounts"
```

deploy_zoom_update.sh

```
- # VERSION: 1.2.0
+ # VERSION: 1.2.1
```

README.md

```
- Version 1.2.0
+ Version 1.2.1
```

CHANGELOG.md

```
+ ## [1.2.1] - 2026-07-07
+ - Fixed privilege check on non-admin accounts
```

build-info.plist (munkipkg)

```
- <string>1.2.0</string>
+ <string>1.2.1</string>
```

> **The package, the script header, the README, and the changelog. All the same version. Always.**

Shikomi adapts to where you run it

INSIDE AN EXISTING REPO

Monorepo mode

```
$ cd ~/jamf-scripts
$ shikomi inventory_collector
```

- Files created in current directory
- Namespaced to avoid collisions
- Offers to create a feature branch

OUTSIDE ANY REPO

Micro-repo mode

```
$ cd ~/Desktop
$ shikomi standalone_utility
```

- Creates a new directory
- Initializes a Git repo
- Optionally creates and pushes the GitHub repo

Shikomi detects which mode applies; you don't have to choose.

You don't have to migrate. You have to start.

1. **For your next new script: scaffold with Shikomi**

```
$ shikomi my_next_script
```

2. **For an existing repo full of scripts: add the hooks without touching the scripts**

```
$ add_security_tools.sh
```

3. **For an existing script that has no version: initialize without changing logic**

```
$ bump-version init
```

> **Do not migrate fifty legacy scripts at once. Let the next new one prove the workflow.**

PREREQUISITES

```
$ brew install pre-commit gitleaks # required
$ brew install gh 1password-cli # optional
```

Three pillars

01

Version control

One canonical version. Full change history. Reviewable diffs across the fleet.

02

Secret scanning

Gitleaks blocks credentials at commit and in CI. Two gates, not one.

03

Automated linting

ShellCheck catches the bugs that fail silently on one machine and look like hardware issues.

You don’t have to use Shikomi.

You should be doing all of this. The tools exist. Now you have no excuse.

Get started

REPOSITORY

github.com/maparker/shikomi

MIT licensed · full docs in the README

RETROFIT EXISTING REPOS

```
$ add_security_tools.sh
```

ISSUES, IDEAS, PRS

github.com/maparker/shikomi/issues

Questions?



MACADMINS SLACK

@mattparker