

amux - Agent modified tmux

Overview

amux (*Agent-Modified tmux*) is a custom build of tmux designed to provide clean, robust bidirectional programmatic control for AI CLI agents and automated scripting.

Standard tmux requires emulating physical keystrokes (`send-keys`) and screen-scraping the visual terminal grid (`capture-pane`). This introduces significant failure modes:

1. **Quoting and Escaping Hell:** Nested quotes, shell variables, subshells, and escapes are easily mangled when passed across nested SSH sessions or bash wrappers.
2. **Terminal Echo & Input Race Conditions:** Rapid key injection can race with shell readline redraws, bracketed paste triggers, or auto-suggestions.
3. **ANSI & Layout Pollution:** Screen scraping mixes prompt redraws, color escape codes, truncated lines at terminal column boundaries, and terminal dimensions with actual command output.
4. **No Command Completion or Busy-State Visibility:** Standard tmux cannot distinguish a pane running a command from a pane sitting idle at a prompt without polling `capture-pane` or guessing via `monitor-silence`.

amux provides symmetric, full-duplex programmatic control:

- **Zero-Escape Writing (`write-pane -f`):** Direct PTY stream injection bypassing keystroke emulation and shell escaping, with immediate `PANE_CMDRUNNING` state pre-marking on execution (`-e` or trailing `\n`).
- **Direct Stream Reading & Synchronous Waiting (`read-pane`):** Direct PTY output buffer capture (`wp->stream_buf`) bypassing 2D screen grid layout truncation and ANSI pollution, with deterministic OSC 133 / `tcgetpgrp` command completion waiting (`-w`), state inspection (`-S`), and exit-code propagation (`-e` / `-E`).

Architecture & Session Hierarchy

The tmux sessions on the host are organized into clean layers managed by

```
/home/luke/bin/tmux_control:
```

```

+-----+
|  mtx / mx: Outer Master Session                               |
|  Socket: /tmp/tmux-1000/tmux_luke_mtx                         |
|  Config: ~/.tmux.conf_outer                                   |
|                                                                |
|  +-----+                                                  |
|  |  stx / sx: User Interactive Session                       |  |
|  |  Socket: /tmp/tmux-1000/tmux_luke_stx                     |  |
|  |  Config: ~/.tmux.conf_inner                               |  |
|  +-----+                                                  |
|                                                                |
|  +-----+                                                  |
|  |  atx / ax: Agent Session (amux)                           |  |
|  |  Socket: /tmp/tmux-1000/tmux_luke_atx                     |  |
|  |  Config: ~/.tmux.conf_agent                               |  |
|  +-----+                                                  |
+-----+

```

- **atx [session]**: Attaches to or creates the agent session inside `mtx`. Styled with a high-visibility orange status bar (`#ff8800`), `[AGENT]` badge, live per-window `[RUN]` and `[ERR:<code>]` badges, and `C-b` prefix key (matching inner session convention without colliding with outer `mtx` `C-a`).
- **ax <command>**: Runs commands directly on the agent socket (e.g. `ax list-panes`, `ax write-pane ...`, `ax read-pane ...`).
- **Binary**: The custom binary is built from `/home/luke/git/amux` (`/home/luke/git/tmux`) and installed at `/usr/local/bin/tmux-custom`. The system `tmux` (`/usr/bin/tmux`) remains unmodified as a fallback.

Persistent Agent Environment & OSC 133 Shell Integration

When the agent session spawns, it automatically sources `~/.agent_profile`:

- **Persistent Environment Variables:**
 - `AGENT_SESSION=1`
 - `ROUTER=danknasty`
 - `ROUTER_REPO=/home/luke/git/router_config`
 - `MC_DIR=/data/minecraft`

- `MC_ROUTER_V4=10.0.0.113`
- `MC_ROUTER_V6=fd77:7777:5010::1337:f1ac:d00d:3c`
- `MC_BACKEND_PREFIX=fd77:7777:5010::1337:3c:d00d:`

- **Persistent Helper Functions:**

- `rcfg`: Git wrapper for router config (`git -C $ROUTER_REPO`)
- `rssh`: Direct SSH to danknasty
- `rdo <cmd>`: Direct non-interactive execution on router
- `mipctl`: Shortcut for `/data/minecraft/ipctl`

- **OSC 133 Semantic Prompt & Command Hooks:**

- `PS0` emits `\033]133;C;k=1\007` immediately before command execution begins.
- `PROMPT_COMMAND` captures the command's exit status `$?` (`__amux_osc133_pre`) before history syncing and emits `\033]133;D;<exit_code>\007\033]133;A\007` (`__amux_osc133_post`) right before rendering `PS1` (`\033]133;B\007`).

Window Naming & Task Lifecycle Methodology

To maintain strict isolation between tasks, prevent command collisions, and keep workspaces clean, windows in `atx` follow a structured naming convention:

```
<project_or_domain>-<target>-<task_suffix>
```

Naming Pattern Examples

- `mc-pokelovers-blumap`: BlueMap setup or rendering for pokelovers
- `mc-pokelovers-datapacks`: Managing datapacks and gameplay tweaks
- `mc-pokelovers-console`: Direct interactive Minecraft server console
- `rtr-dns-sync`: Router DNS record and deSEC synchronization
- `rtr-pf-rules`: Firewall rule adjustments and testing
- `bld-amux-stream`: Compiling and patching tmux C source code
- `bkp-verify-daily`: Backup execution and PAR2 verification

Window Lifecycle Protocol

1. **Window Creation (Task Start):** Before running multi-step commands for a task, verify if the dedicated window exists; if not, spawn it:

```
ax list-windows -F '#{window_name}' | grep -q '^mc-pokelovers-bluemap$' || \  
ax new-window -n "mc-pokelovers-bluemap" "bash --rcfile ~/.agent_profile"
```

2. **Scoped Execution via File Injection:** Write the script or commands to a file on `/dev/shm`, flush the output buffer, and inject into the target window using `write-pane -f`:

```
ax read-pane -C -t "mc-pokelovers-bluemap"  
ax write-pane -t "mc-pokelovers-bluemap" -e -f /dev/shm/task.sh  
rm -f /dev/shm/task.sh
```

3. **Synchronous Output Retrieval & Exit Verification:** Wait for the command or multi-line script to finish, isolate the clean OSC 133 output, drain the queue, and propagate the exit code:

```
ax read-pane -w -0 -c -E -t "mc-pokelovers-bluemap"
```

4. **Status Renaming:** If a task transitions to a persistent state (e.g. watching logs, running tests), update the title:

```
ax rename-window -t "mc-pokelovers-bluemap" "mc-pokelovers-watch"
```

5. **Task Completion Confirmation & Cleanup Rule:**

- When a task or subtask is complete, the outcome must be explicitly verified (service status, connectivity, file state).
- Once verified, promptly close the dedicated task window:

```
ax kill-window -t "mc-pokelovers-bluemap"
```

- No completed task windows are left lingering in the agent session. Confirm to the user that the task is finished and that its task window has been torn down.

Built-In Commands

`write-pane` (alias: `writep`)

Injects raw binary data, files, or Base64 payloads directly into the target pane's PTY input buffer. It completely bypasses terminal keystroke decoding and shell quoting, records the current stream buffer mark (`wp->stream_mark`), and immediately pre-marks the pane as `running` (`PANE_CMDRUNNING`) when newline execution (`-e` or trailing `\n`) is triggered.

```
tmux write-pane [-en] [-f file] [-b base64-data] [-t target-pane] [data ...]
```

Flags:

- **-f <file>**: Reads the given file from disk and streams its raw bytes directly into the pane's input buffer. **(Recommended Standard)**
- **-e**: Automatically appends a newline (`\n` / Enter) to trigger immediate execution.
- **-b <base64-data>**: Decodes a base64 payload in C (`b64_pton`) and writes raw bytes to the PTY.
- **-n**: Do not separate multiple positional arguments with spaces.
- **-t <target-pane>**: Specifies the target session, window, or pane (e.g. `-t mc-poke lovers-blue map`).

read-pane (alias: readp)

Reads raw or filtered output directly from the target pane's 16 MB in-memory stream buffer (`wp->stream_buf`), queries command execution state/exit status, or synchronously waits for command completion without screen scraping or polling.

```
tmux read-pane [-cCeEl0prSsw] [-b buffer-name] [-f file] [-o file] [-T timeout] [-t target-pane]
```

Flags:

- **Default (Clean Mode)**: Automatically strips ANSI / VT100 CSI, OSC, APC/title, and charset escape sequences, removes bracketed paste triggers, and normalizes `\r\n` linebreaks to standard `\n`.
- **-w (wait for completion)**: Suspends the client until all active/queued commands in the target pane finish (`OSC 133;D` / `133;A` with a 25 ms multi-line settle window, or PTY foreground process group completion) before reading.
- **-T <seconds> (timeout)**: Maximum seconds to wait when `-w` is specified; exits with return code `124` on timeout.
- **-0 (OSC 133 isolation)**: Semantically extracts command output blocks between `OSC 133;C` (output start) and `OSC 133;D` (output end) produced since the last `write-pane` (or buffer drain), completely eliminating prompt redraws and command echo across both single-line and multi-line script injections.
- **-c (consume)**: Drains read bytes from the pane buffer (FIFO behavior). Repeated calls return only *new* output produced since the last read.
- **-C (clear/flush)**: Immediately flushes the pane output buffer without reading (ideal before running a new command).
- **-S (status summary)**: Prints a single-line structured state summary without modifying the stream buffer:

```
state=<idle|running|dead|unknown> exit_code=<N|-> duration=<N> unread_bytes=<N>
command=<cmd>
```

- **-e (exit code):** Prints the numeric exit status (`0..255` , or `-1` if running/unknown) of the last completed OSC 133 command.
- **-E (propagate exit code):** Propagates a non-zero command exit status from the target pane directly to the `read-pane` client exit status (`$?`).
- **-l (length):** Returns the current number of queued bytes in the pane buffer without draining.
- **-r (raw):** Bypasses ANSI stripping and outputs the raw, unmodified PTY byte stream.
- **-s (strip):** Explicitly strip ANSI escape codes (default behavior).
- **-o <file> / -f <file>:** Streams buffer content directly to a file on disk.
- **-b <buffer-name>:** Stores captured content into a tmux paste buffer.
- **-t <target-pane>:** Target pane specification.

Format Variables & Status Bar Integration

amux exposes pane command state and buffer metrics as native tmux format variables usable in `list-panes -F` , `display-message -p` , and status lines:

Format Variable	Description
<code>{pane_command_state}</code>	<code>running</code> , <code>idle</code> , <code>dead</code> , or <code>unknown</code> (combines OSC 133 and PTY <code>tcgetpgrp</code> tracking)
<code>{pane_last_exit_code}</code>	Exit code (<code>0..255</code>) of the last completed OSC 133 command (empty if none)
<code>{pane_stream_bytes}</code>	Unread byte count currently queued in <code>wp->stream_buf</code>
<code>{pane_command_running}</code>	<code>1</code> if an OSC 133 command is running, <code>0</code> otherwise
<code>{pane_command_status}</code>	Exit status from most recent OSC 133 command
<code>{pane_command_duration}</code>	Elapsed duration in seconds of the current or most recent OSC 133 command

Inspecting All Agent Panes at Once

```
ax list-panes -a -F '#{window_name} ({pane_id}): state=#{pane_command_state}
exit=#{pane_last_exit_code} duration=#{pane_command_duration}s unread=#{pane_stream_bytes}B'
```

In `~/.tmux.conf_agent` , window titles in the orange `[AGENT]` status bar automatically display `[RUN]` while a command is executing and `[ERR:<code>]` if the last command exited with a non-zero code.

Standard Workflow: Synchronous Zero-Escape Bidirectional Automation

Step 1: Write Script to `/dev/shm`

```
cat > /dev/shm/job.sh << 'SCRIPT'
python3 - << 'PY'
import json, os
print(json.dumps({
    "user": os.getenv("USER"),
    "quote_test": 'Double "quotes" and single \'quotes\' work without escapes!'
}, indent=2))
PY
SCRIPT
```

Step 2: Flush, Inject, Wait, and Read Clean Output

```
ax read-pane -C -t mc-pokelovers-datapacks
ax write-pane -t mc-pokelovers-datapacks -e -f /dev/shm/job.sh
rm -f /dev/shm/job.sh

# Blocks until completion, extracts clean OSC 133 output, drains buffer, and propagates exit
code
ax read-pane -w -0 -c -E -t mc-pokelovers-datapacks
```

Structured Audit Logging

Both `write-pane` and `read-pane` integrate with systemd journal for structured security and execution logging:

- Injected scripts are archived to `/data/amux/scripts/<timestamp>_<window>.sh`.
- Audit events are sent to the systemd journal under identifier `amux` (including `AMUX_PANE_ID`, `AMUX_WINDOW_NAME`, `AMUX_READ_SIZE`, `AMUX_CONSUMED`, `AMUX_CMD_STATE`, and `AMUX_EXIT_CODE`):

```
journalctl -t amux -n 20 --no-pager
```

Rebuilding & Updating amux

The source tree is maintained at `/home/luke/git/amux` (`/home/luke/git/tmux`):

```
cd /home/luke/git/amux
make -j"$(nproc)"
sudo install -m 755 tmux /usr/local/bin/tmux-custom
```

Existing sessions can continue running undisturbed; new sessions or restarts of `tmux_control` (`atx`, `stx`, `mtx`) automatically pick up the updated binary.

Revision #9

Created 2026-09-27 22:10:33 UTC by luKe

Updated 2026-10-19 11:09:45 UTC by luKe