agent-browser

agent-browser

Headless browser automation CLI for AI agents. Fast Rust CLI with Node.js fallback.

Installation

npm (recommended)

```
npm install -g agent-browser
agent-browser install  # Download Chromium
```

From Source

```
git clone https://github.com/vercel-labs/agent-browser
cd agent-browser
pnpm install
pnpm build
pnpm build:native   # Requires Rust (https://rustup.rs)
pnpm link --global  # Makes agent-browser available globally
agent-browser install
```

Linux Dependencies

On Linux, install system dependencies:

```
agent-browser install --with-deps
# or manually: npx playwright install-deps chromium
```

Quick Start

```
agent-browser open example.com
agent-browser snapshot                    # Get accessibility tree with refs
agent-browser click @e2                   # Click by ref from snapshot
agent-browser fill @e3 "test@example.com" # Fill by ref
agent-browser get text @e1                # Get text by ref
agent-browser screenshot page.png
agent-browser close
```

Traditional Selectors (also supported)

```
agent-browser click "#submit"
agent-browser fill "#email" "test@example.com"
agent-browser find role button click --name "Submit"
```

Commands

Core Commands

```
agent-browser open <url>              # Navigate to URL (aliases: goto, navigate)
agent-browser click <sel>             # Click element
agent-browser dblclick <sel>          # Double-click element
agent-browser focus <sel>             # Focus element
agent-browser type <sel> <text>       # Type into element
agent-browser fill <sel> <text>       # Clear and fill
agent-browser press <key>             # Press key (Enter, Tab, Control+a) (alias: key)
agent-browser keydown <key>           # Hold key down
agent-browser keyup <key>             # Release key
agent-browser hover <sel>             # Hover element
agent-browser select <sel> <val>      # Select dropdown option
agent-browser check <sel>             # Check checkbox
agent-browser uncheck <sel>           # Uncheck checkbox
agent-browser scroll <dir> [px]       # Scroll (up/down/left/right)
agent-browser scrollintoview <sel>    # Scroll element into view (alias: scrollinto)
agent-browser drag <src> <tgt>        # Drag and drop
agent-browser upload <sel> <files>    # Upload files
agent-browser screenshot [path]       # Take screenshot (--full for full page, saves to a temporary directory if no path)
agent-browser pdf <path>              # Save as PDF
agent-browser snapshot                # Accessibility tree with refs (best for AI)
agent-browser eval <js>               # Run JavaScript
agent-browser connect <port>          # Connect to browser via CDP
agent-browser close                   # Close browser (aliases: quit, exit)
```

Get Info

```
agent-browser get text <sel>          # Get text content
agent-browser get html <sel>          # Get innerHTML
agent-browser get value <sel>         # Get input value
```

```
agent-browser get attr <sel> <attr>    # Get attribute
agent-browser get title                # Get page title
agent-browser get url                  # Get current URL
agent-browser get count <sel>          # Count matching elements
agent-browser get box <sel>            # Get bounding box
```

Check State

```
agent-browser is visible <sel>         # Check if visible
agent-browser is enabled <sel>         # Check if enabled
agent-browser is checked <sel>         # Check if checked
```

Find Elements (Semantic Locators)

```
agent-browser find role <role> <action> [value]        # By ARIA role
agent-browser find text <text> <action>                # By text content
agent-browser find label <label> <action> [value]      # By label
agent-browser find placeholder <ph> <action> [value]   # By placeholder
agent-browser find alt <text> <action>                 # By alt text
agent-browser find title <text> <action>               # By title attr
agent-browser find testid <id> <action> [value]        # By data-testid
agent-browser find first <sel> <action> [value]        # First match
agent-browser find last <sel> <action> [value]         # Last match
agent-browser find nth <n> <sel> <action> [value]      # Nth match
```

Actions: `click`, `fill`, `check`, `hover`, `text`

Examples:

```
agent-browser find role button click --name "Submit"
agent-browser find text "Sign In" click
agent-browser find label "Email" fill "test@test.com"
agent-browser find first ".item" click
agent-browser find nth 2 "a" text
```

Wait

```
agent-browser wait <selector>          # Wait for element to be visible
agent-browser wait <ms>                # Wait for time (milliseconds)
agent-browser wait --text "Welcome"    # Wait for text to appear
agent-browser wait --url "**/dash"     # Wait for URL pattern
```

```
agent-browser wait --load networkidle # Wait for load state
agent-browser wait --fn "window.ready === true"  # Wait for JS condition
```

Load states: `load`, `domcontentloaded`, `networkidle`

Mouse Control

```
agent-browser mouse move <x> <y>       # Move mouse
agent-browser mouse down [button]      # Press button (left/right/middle)
agent-browser mouse up [button]        # Release button
agent-browser mouse wheel <dy> [dx]    # Scroll wheel
```

Browser Settings

```
agent-browser set viewport <w> <h>     # Set viewport size
agent-browser set device <name>        # Emulate device ("iPhone 14")
agent-browser set geo <lat> <lng>      # Set geolocation
agent-browser set offline [on|off]     # Toggle offline mode
agent-browser set headers <json>       # Extra HTTP headers
agent-browser set credentials <u> <p> # HTTP basic auth
agent-browser set media [dark|light]   # Emulate color scheme
```

Cookies & Storage

```
agent-browser cookies                  # Get all cookies
agent-browser cookies set <name> <val> # Set cookie
agent-browser cookies clear            # Clear cookies

agent-browser storage local            # Get all localStorage
agent-browser storage local <key>      # Get specific key
agent-browser storage local set <k> <v>  # Set value
agent-browser storage local clear      # Clear all

agent-browser storage session          # Same for sessionStorage
```

Network

```
agent-browser network route <url>              # Intercept requests
agent-browser network route <url> --abort      # Block requests
agent-browser network route <url> --body <json>  # Mock response
agent-browser network unroute [url]            # Remove routes
```

```
agent-browser network requests                 # View tracked requests
agent-browser network requests --filter api    # Filter requests
```

Tabs & Windows

```
agent-browser tab                     # List tabs
agent-browser tab new [url]           # New tab (optionally with URL)
agent-browser tab <n>                 # Switch to tab n
agent-browser tab close [n]           # Close tab
agent-browser window new              # New window
```

Frames

```
agent-browser frame <sel>             # Switch to iframe
agent-browser frame main              # Back to main frame
```

Dialogs

```
agent-browser dialog accept [text]    # Accept (with optional prompt text)
agent-browser dialog dismiss          # Dismiss
```

Debug

```
agent-browser trace start [path]      # Start recording trace
agent-browser trace stop [path]       # Stop and save trace
agent-browser console                 # View console messages (log, error,
warn, info)
agent-browser console --clear         # Clear console
agent-browser errors                  # View page errors (uncaught JavaScript
exceptions)
agent-browser errors --clear          # Clear errors
agent-browser highlight <sel>         # Highlight element
agent-browser state save <path>       # Save auth state
agent-browser state load <path>       # Load auth state
```

Navigation

```
agent-browser back                    # Go back
agent-browser forward                 # Go forward
agent-browser reload                  # Reload page
```

Setup

```
agent-browser install                # Download Chromium browser
agent-browser install --with-deps    # Also install system deps (Linux)
```

Sessions

Run multiple isolated browser instances:

```
# Different sessions
agent-browser --session agent1 open site-a.com
agent-browser --session agent2 open site-b.com

# Or via environment variable
AGENT_BROWSER_SESSION=agent1 agent-browser click "#btn"

# List active sessions
agent-browser session list
# Output:
# Active sessions:
# -> default
#    agent1

# Show current session
agent-browser session
```

Each session has its own:

- Browser instance
- Cookies and storage
- Navigation history
- Authentication state

Persistent Profiles

By default, browser state (cookies, localStorage, login sessions) is ephemeral and lost when the browser closes. Use `--profile` to persist state across browser restarts:

```
# Use a persistent profile directory
agent-browser --profile ~/.myapp-profile open myapp.com

# Login once, then reuse the authenticated session
agent-browser --profile ~/.myapp-profile open myapp.com/dashboard
```

```
# Or via environment variable
AGENT_BROWSER_PROFILE=~/.myapp-profile agent-browser open myapp.com
```

The profile directory stores:

- Cookies and localStorage
- IndexedDB data
- Service workers
- Browser cache
- Login sessions

Tip: Use different profile paths for different projects to keep their browser state isolated.

Snapshot Options

The `snapshot` command supports filtering to reduce output size:

```
agent-browser snapshot                     # Full accessibility tree
agent-browser snapshot -i                  # Interactive elements only
(buttons, inputs, links)
agent-browser snapshot -c                  # Compact (remove empty structural
elements)
agent-browser snapshot -d 3                # Limit depth to 3 levels
agent-browser snapshot -s "#main"          # Scope to CSS selector
agent-browser snapshot -i -c -d 5          # Combine options
```

Option	Description
`-i, --interactive`	Only show interactive elements (buttons, links, inputs)
`-c, --compact`	Remove empty structural elements
`-d, --depth <n>`	Limit tree depth
`-s, --selector <sel>`	Scope to CSS selector

Options

Option	Description
`--session <name>`	Use isolated session (or `AGENT_BROWSER_SESSION` env)
`--profile <path>`	Persistent browser profile directory (or `AGENT_BROWSER_PROFILE` env)

Option	Description
`--headers <json>`	Set HTTP headers scoped to the URL's origin
`--executable-path <path>`	Custom browser executable (or `AGENT_BROWSER_EXECUTABLE_PATH` env)
`--args <args>`	Browser launch args, comma or newline separated (or `AGENT_BROWSER_ARGS` env)
`--user-agent <ua>`	Custom User-Agent string (or `AGENT_BROWSER_USER_AGENT` env)
`--proxy <url>`	Proxy server URL with optional auth (or `AGENT_BROWSER_PROXY` env)
`--proxy-bypass <hosts>`	Hosts to bypass proxy (or `AGENT_BROWSER_PROXY_BYPASS` env)
`-p, --provider <name>`	Cloud browser provider (or `AGENT_BROWSER_PROVIDER` env)
`--json`	JSON output (for agents)
`--full, -f`	Full page screenshot
`--name, -n`	Locator name filter
`--exact`	Exact text match
`--headed`	Show browser window (not headless)
`--cdp <port>`	Connect via Chrome DevTools Protocol
`--ignore-https-errors`	Ignore HTTPS certificate errors (useful for self-signed certs)
`--debug`	Debug output

Selectors

Refs (Recommended for AI)

Refs provide deterministic element selection from snapshots:

```
# 1. Get snapshot with refs
agent-browser snapshot
# Output:
# - heading "Example Domain" [ref=e1] [level=1]
# - button "Submit" [ref=e2]
# - textbox "Email" [ref=e3]
# - link "Learn more" [ref=e4]

# 2. Use refs to interact
agent-browser click @e2                   # Click the button
```

```
agent-browser fill @e3 "test@example.com" # Fill the textbox
agent-browser get text @e1                # Get heading text
agent-browser hover @e4                   # Hover the link
```

Why use refs?

- **Deterministic**: Ref points to exact element from snapshot
- **Fast**: No DOM re-query needed
- **AI-friendly**: Snapshot + ref workflow is optimal for LLMs

CSS Selectors

```
agent-browser click "#id"
agent-browser click ".class"
agent-browser click "div > button"
```

Text & XPath

```
agent-browser click "text=Submit"
agent-browser click "xpath=//button"
```

Semantic Locators

```
agent-browser find role button click --name "Submit"
agent-browser find label "Email" fill "test@test.com"
```

Agent Mode

Use `--json` for machine-readable output:

```
agent-browser snapshot --json
# Returns: {"success":true,"data":{"snapshot":"...","refs":{"e1":
{"role":"heading","name":"Title"},...}}}

agent-browser get text @e1 --json
agent-browser is visible @e2 --json
```

Optimal AI Workflow

```
# 1. Navigate and get snapshot
agent-browser open example.com
```

```
agent-browser snapshot -i --json   # AI parses tree and refs

# 2. AI identifies target refs from snapshot
# 3. Execute actions using refs
agent-browser click @e2
agent-browser fill @e3 "input text"

# 4. Get new snapshot if page changed
agent-browser snapshot -i --json
```

Headed Mode

Show the browser window for debugging:

```
agent-browser open example.com --headed
```

This opens a visible browser window instead of running headless.

Authenticated Sessions

Use `--headers` to set HTTP headers for a specific origin, enabling authentication without login flows:

```
# Headers are scoped to api.example.com only
agent-browser open api.example.com --headers '{"Authorization": "Bearer
<token>"}'

# Requests to api.example.com include the auth header
agent-browser snapshot -i --json
agent-browser click @e2

# Navigate to another domain - headers are NOT sent (safe!)
agent-browser open other-site.com
```

This is useful for:

- **Skipping login flows** - Authenticate via headers instead of UI
- **Switching users** - Start new sessions with different auth tokens
- **API testing** - Access protected endpoints directly
- **Security** - Headers are scoped to the origin, not leaked to other domains

To set headers for multiple origins, use `--headers` with each `open` command:

```
agent-browser open api.example.com --headers '{"Authorization": "Bearer
token1"}'
agent-browser open api.acme.com --headers '{"Authorization": "Bearer token2"}'
```

For global headers (all domains), use `set headers`:

```
agent-browser set headers '{"X-Custom-Header": "value"}'
```

Custom Browser Executable

Use a custom browser executable instead of the bundled Chromium. This is useful for:

- **Serverless deployment**: Use lightweight Chromium builds like `@sparticuz/chromium` (~50MB vs ~684MB)
- **System browsers**: Use an existing Chrome/Chromium installation
- **Custom builds**: Use modified browser builds

CLI Usage

```
# Via flag
agent-browser --executable-path /path/to/chromium open example.com

# Via environment variable
AGENT_BROWSER_EXECUTABLE_PATH=/path/to/chromium agent-browser open example.com
```

Serverless Example (Vercel/AWS Lambda)

```
import chromium from '@sparticuz/chromium';
import { BrowserManager } from 'agent-browser';

export async function handler() {
  const browser = new BrowserManager();
  await browser.launch({
    executablePath: await chromium.executablePath(),
    headless: true,
  });
  // ... use browser
}
```

CDP Mode

Connect to an existing browser via Chrome DevTools Protocol:

```
# Start Chrome with: google-chrome --remote-debugging-port=9222

# Connect once, then run commands without --cdp
agent-browser connect 9222
agent-browser snapshot
agent-browser tab
agent-browser close

# Or pass --cdp on each command
agent-browser --cdp 9222 snapshot

# Connect to remote browser via WebSocket URL
agent-browser --cdp "wss://your-browser-service.com/cdp?token=..." snapshot
```

The `--cdp` flag accepts either:

- A port number (e.g., `9222`) for local connections via `http://localhost:{port}`
- A full WebSocket URL (e.g., `wss://...` or `ws://...`) for remote browser services

This enables control of:

- Electron apps
- Chrome/Chromium instances with remote debugging
- WebView2 applications
- Any browser exposing a CDP endpoint

Streaming (Browser Preview)

Stream the browser viewport via WebSocket for live preview or "pair browsing" where a human can watch and interact alongside an AI agent.

Enable Streaming

Set the `AGENT_BROWSER_STREAM_PORT` environment variable:

```
AGENT_BROWSER_STREAM_PORT=9223 agent-browser open example.com
```

This starts a WebSocket server on the specified port that streams the browser viewport and accepts input events.

WebSocket Protocol

Connect to `ws://localhost:9223` to receive frames and send input:

Receive frames:

```json
{
  "type": "frame",
  "data": "<base64-encoded-jpeg>",
  "metadata": {
    "deviceWidth": 1280,
    "deviceHeight": 720,
    "pageScaleFactor": 1,
    "offsetTop": 0,
    "scrollOffsetX": 0,
    "scrollOffsetY": 0
  }
}
```

Send mouse events:

```json
{
  "type": "input_mouse",
  "eventType": "mousePressed",
  "x": 100,
  "y": 200,
  "button": "left",
  "clickCount": 1
}
```

Send keyboard events:

```json
{
  "type": "input_keyboard",
  "eventType": "keyDown",
  "key": "Enter",
  "code": "Enter"
}
```

Send touch events:

```json
{
  "type": "input_touch",
  "eventType": "touchStart",
  "touchPoints": [{ "x": 100, "y": 200 }]
}
```

Programmatic API

For advanced use, control streaming directly via the protocol:

```
import { BrowserManager } from 'agent-browser';

const browser = new BrowserManager();
await browser.launch({ headless: true });
await browser.navigate('https://example.com');

// Start screencast
await browser.startScreencast((frame) => {
  // frame.data is base64-encoded image
  // frame.metadata contains viewport info
  console.log('Frame received:', frame.metadata.deviceWidth, 'x',
frame.metadata.deviceHeight);
}, {
  format: 'jpeg',
  quality: 80,
  maxWidth: 1280,
  maxHeight: 720,
});

// Inject mouse events
await browser.injectMouseEvent({
  type: 'mousePressed',
  x: 100,
  y: 200,
  button: 'left',
});

// Inject keyboard events
await browser.injectKeyboardEvent({
  type: 'keyDown',
  key: 'Enter',
  code: 'Enter',
});

// Stop when done
await browser.stopScreencast();
```

Architecture

agent-browser uses a client-daemon architecture:

1. **Rust CLI** (fast native binary) - Parses commands, communicates with daemon
2. **Node.js Daemon** - Manages Playwright browser instance
3. **Fallback** - If native binary unavailable, uses Node.js directly

The daemon starts automatically on first command and persists between commands for fast subsequent operations.

Browser Engine: Uses Chromium by default. The daemon also supports Firefox and WebKit via the Playwright protocol.

Platforms

Platform	Binary	Fallback
macOS ARM64	Native Rust	Node.js
macOS x64	Native Rust	Node.js
Linux ARM64	Native Rust	Node.js
Linux x64	Native Rust	Node.js
Windows x64	Native Rust	Node.js

Usage with AI Agents

Just ask the agent

The simplest approach - just tell your agent to use it:

```
Use agent-browser to test the login flow. Run agent-browser --help to see
available commands.
```

The `--help` output is comprehensive and most agents can figure it out from there.

AI Coding Assistants

Add the skill to your AI coding assistant for richer context:

```
npx skills add vercel-labs/agent-browser
```

This works with Claude Code, Codex, Cursor, Gemini CLI, GitHub Copilot, Goose, OpenCode, and Windsurf.

AGENTS.md / CLAUDE.md

For more consistent results, add to your project or global instructions file:

```
## Browser Automation

Use `agent-browser` for web automation. Run `agent-browser --help` for all
commands.

Core workflow:
1. `agent-browser open <url>` - Navigate to page
2. `agent-browser snapshot -i` - Get interactive elements with refs (@e1, @e2)
3. `agent-browser click @e1` / `fill @e2 "text"` - Interact using refs
4. Re-snapshot after page changes
```

Integrations

Browserbase

Browserbase provides remote browser infrastructure to make deployment of agentic browsing agents easy. Use it when running the agent-browser CLI in an environment where a local browser isn't feasible.

To enable Browserbase, use the `-p` flag:

```
export BROWSERBASE_API_KEY="your-api-key"
export BROWSERBASE_PROJECT_ID="your-project-id"
agent-browser -p browserbase open https://example.com
```

Or use environment variables for CI/scripts:

```
export AGENT_BROWSER_PROVIDER=browserbase
export BROWSERBASE_API_KEY="your-api-key"
export BROWSERBASE_PROJECT_ID="your-project-id"
agent-browser open https://example.com
```

When enabled, agent-browser connects to a Browserbase session instead of launching a local browser. All commands work identically.

Get your API key and project ID from the Browserbase Dashboard.

Browser Use

Browser Use provides cloud browser infrastructure for AI agents. Use it when running agent-browser in environments where a local browser isn't available (serverless, CI/CD, etc.).

To enable Browser Use, use the `-p` flag:

```
export BROWSER_USE_API_KEY="your-api-key"
agent-browser -p browseruse open https://example.com
```

Or use environment variables for CI/scripts:

```
export AGENT_BROWSER_PROVIDER=browseruse
export BROWSER_USE_API_KEY="your-api-key"
agent-browser open https://example.com
```

When enabled, agent-browser connects to a Browser Use cloud session instead of launching a local browser. All commands work identically.

Get your API key from the Browser Use Cloud Dashboard. Free credits are available to get started, with pay-as-you-go pricing after.

Kernel

Kernel provides cloud browser infrastructure for AI agents with features like stealth mode and persistent profiles.

To enable Kernel, use the `-p` flag:

```
export KERNEL_API_KEY="your-api-key"
agent-browser -p kernel open https://example.com
```

Or use environment variables for CI/scripts:

```
export AGENT_BROWSER_PROVIDER=kernel
export KERNEL_API_KEY="your-api-key"
agent-browser open https://example.com
```

Optional configuration via environment variables:

Variable	Description	Default
`KERNEL_HEADLESS`	Run browser in headless mode (`true` / `false`)	`false`
`KERNEL_STEALTH`	Enable stealth mode to avoid bot detection (`true` / `false`)	`true`
`KERNEL_TIMEOUT_SECONDS`	Session timeout in seconds	`300`

Variable	Description	Default
`KERNEL_PROFILE_NAME`	Browser profile name for persistent cookies/logins (created if it doesn't exist)	(none)

When enabled, agent-browser connects to a Kernel cloud session instead of launching a local browser. All commands work identically.

Profile Persistence: When `KERNEL_PROFILE_NAME` is set, the profile will be created if it doesn't already exist. Cookies, logins, and session data are automatically saved back to the profile when the browser session ends, making them available for future sessions.

Get your API key from the [Kernel Dashboard].

License

Apache-2.0