1 Branch 1 Tag Go to file Go to file Code

cosinusalpha Bump version to 0.1.2 in… ✓

353b481 · yesterday

.github/w…	Enhance CI an…	2 days ago
src/webctl	Enhance Daem…	2 days ago
tests	Refactor smoke…	2 days ago
.gitignore	Enhance conso…	2 days ago
.python-…	Refactor code s…	2 days ago
READM…	chg: improved …	yesterday
pyproject…	Bump version t…	yesterday
uv.lock	Refactor code s…	2 days ago

About

Browser automation via CLI — for humans and agents

Readme
Activity
139 stars
0 watching
6 forks
Report repository

Releases

1 tags

Packages

No packages published

README

Python 100.0%

webctl

Browser automation for AI agents and humans, built on the command line.

```
webctl start
webctl navigate "https://google.com"
webctl type 'role=combobox name~="Search"' "best restaurants nearby" --submit
webctl snapshot --interactive-only --limit 20
webctl stop --daemon
```

Why CLI Instead of MCP?

MCP browser tools have a fundamental problem: **the server controls what enters your context**. With Playwright MCP, every response includes the full accessibility tree plus console messages (default: "info" level). After a few page queries, your context is full.

CLI flips this around: **you control what enters context**.

```
# Filter before context
webctl snapshot --interactive-only --limit 30      # Only buttons, links, inputs
webctl snapshot --within "role=main"               # Skip nav, footer, ads

# Pipe through Unix tools
webctl snapshot | grep -i "submit"                 # Find specific elements
webctl --format jsonl snapshot | jq '.data.role'   # Extract with jq
webctl snapshot | head -50                         # Truncate output
```

Beyond filtering, CLI gives you:

Capability	CLI	MCP
Filter output	Built-in flags + grep/jq/head	Server decides
Debug	Run same command as agent	Opaque
Cache	`webctl snapshot > cache.txt`	Every call hits server
Script	Save to .sh, version control	Ephemeral
Timeout	`timeout 30 webctl ...`	Internal only
Parallelize	`parallel` , `xargs` , `&`	Server-dependent
Human takeover	Same commands	Different interface

Quick Start

```
pip install webctl      # Requires Python 3.11+
webctl setup            # Downloads Chromium (~150MB)
```

Verify it works:

```
webctl start
webctl navigate "https://example.com"
webctl snapshot --interactive-only
webctl stop --daemon
```

▶ Install from source

▶ Linux system dependencies

Core Concepts

Sessions

Browser stays open across commands. Cookies persist to disk.

```
webctl start                    # Visible browser
webctl start --mode unattended  # Headless
webctl -s work start            # Named profile (separate cookies)
webctl stop --daemon            # Shutdown everything
```

Element Queries

Semantic targeting based on ARIA roles - stable across CSS refactors:

```
role=button                     # Any button
role=button name="Submit"       # Exact match
role=button name~="Submit"      # Contains (preferred)
role=textbox name~="Email"      # Input field
role=link name~="Sign in"       # Link
```

Output Control

```
webctl snapshot                                    # Human-readable
webctl --quiet navigate "..."                      # Suppress events
webctl --result-only --format jsonl navigate "..." # Pure JSON, final result only
```

Commands

Navigation

```
webctl navigate "https://..."   # Go to URL
webctl back                     # History back
webctl forward                  # History forward
webctl reload                   # Refresh
```

Observation

```
webctl snapshot                          # Full a11y tree
webctl snapshot --interactive-only       # Buttons, links, inputs only
webctl snapshot --limit 30               # Cap output
webctl snapshot --within "role=main"     # Scope to container
webctl snapshot --roles "button,link"    # Filter by role
webctl query "role=button name~=Submit"  # Debug query, get suggestions
webctl screenshot --path shot.png        # Screenshot
```

Interaction

```
webctl click 'role=button name~="Submit"'
webctl type 'role=textbox name~="Email"' "user@example.com"
webctl type 'role=textbox name~="Search"' "query" --submit  # Type + Enter
webctl select 'role=combobox name~="Country"' --label "Germany"
webctl check 'role=checkbox name~="Remember"'
webctl press Enter
webctl scroll down
webctl upload 'role=button name~="Upload"' --file ./doc.pdf
```

Wait Conditions

```
webctl wait network-idle
webctl wait 'exists:role=button name~="Continue"'
webctl wait 'visible:role=dialog'
webctl wait 'hidden:role=progressbar'
webctl wait 'url-contains:"/dashboard"'
```

Session Management

```
webctl status                # Current state (includes console error counts)
webctl save                  # Persist cookies now
webctl sessions              # List profiles
webctl pages                 # List tabs
webctl focus p2              # Switch tab
webctl close-page p1         # Close tab
```

Console Logs

```
webctl console               # Get last 100 logs
webctl console --count       # Just counts by level (LLM-friendly)
webctl console --level error # Filter to errors only
webctl console --follow      # Stream new logs continuously
webctl console -n 50 -l warn # Last 50 warnings
```

Setup & Config

```
webctl setup                    # Install browser
webctl doctor                   # Diagnose installation
webctl init                     # Add to agent configs (CLAUDE.md, etc.)
webctl config show              # Show settings
webctl config set idle_timeout 1800
```

Agent Integration

Tell your AI agent to use webctl. The easiest way:

```
webctl init                     # Creates CLAUDE.md, GEMINI.md, etc.
webctl init --agents claude     # Only specific agents
```

Or manually add to your agent's config:

```
For web browsing, use webctl CLI. Run `webctl agent-prompt` for instructions.
```

For AI Agents

This section is designed to be read by AI agents directly.

webctl Quick Reference

Control a browser via CLI. Start with `webctl start`, end with `webctl stop --daemon`.

Commands:

```
webctl start                            # Open browser
webctl navigate "URL"                   # Go to URL
webctl snapshot --interactive-only      # See clickable elements
webctl click 'role=button name~="Text"'   # Click element
webctl type 'role=textbox name~="Field"' "text"           # Type
webctl type 'role=textbox name~="Field"' "text" --submit  # Type + Enter
webctl select 'role=combobox' --label "Option"            # Dropdown
webctl wait 'exists:role=button name~="..."'              # Wait for element
webctl stop --daemon                    # Close browser
```

Query syntax:

- `role=button` - By ARIA role (button, link, textbox, combobox, checkbox)

- `name~="partial"` - Partial match (preferred, more robust)
- `name="exact"` - Exact match

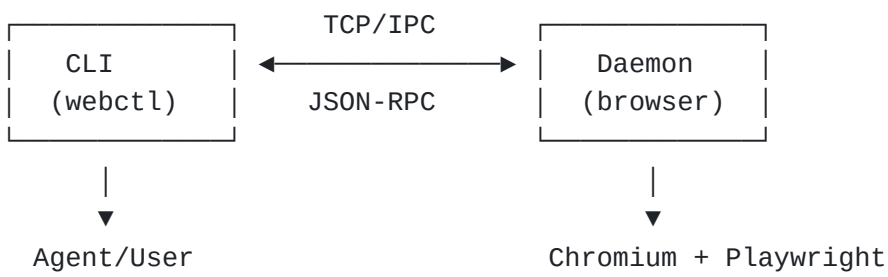
Example - Login:

```
webctl start
webctl navigate "https://site.com/login"
webctl type 'role=textbox name~="Email"' "user@example.com"
webctl type 'role=textbox name~="Password"' "secret" --submit
webctl wait 'url-contains:"/dashboard"'
```

Tips:

- Use `--interactive-only` to reduce output (only buttons, links, inputs)
- Use `name~=` for partial matching (handles minor text changes)
- Use `webctl query "..."` if element not found - shows suggestions
- Use `--quiet` to suppress event output
- Sessions persist cookies - login once, stay logged in
- Check `webctl status` for console error counts before investigating
- Use `webctl console --count` for log summary, `--level error` for details

Architecture



- **CLI**: Stateless, sends commands to daemon
- **Daemon**: Manages browser, auto-starts on first command
- **Profiles**: `~/.local/share/webctl/profiles/`
- **Config**: `~/.config/webctl/config.json`