URSA

Setup Reference Card

Universal Research and Scientific Agent • Docs: lanl.github.io/ursa

URSA is a flexible agentic workflow for scientific tasks. Its terminal interface can chat with a model, invoke specialized planning and execution agents, preserve named agents, and connect to configurable model endpoints. This card covers installing URSA, connecting it to the American Science Cloud Model Access Gateway (MAG), and starting a safe first session.

1 · Install

URSA is published on PyPI as ursa-ai and requires Python 3.11 or newer. uv is recommended. Use a tool install for CLI-only work or a virtual environment when Python scripts must import URSA.

Use case	Install command	Update
CLI only (recommended)	`uv tool install ursa-ai`	uv tool upgrade ursa-ai

uv virtual environment	`uv venv --python 3.12 .venv`
	`source .venv/bin/activate`
	`uv pip install ursa-ai`
		uv pip install --upgrade ursa-ai

pip virtual environment	`python3 -m venv .venv`
	`source .venv/bin/activate`
	`python -m pip install ursa-ai`
		python -m pip install --upgrade ursa-ai

Windows PowerShell activation: .\.venv\Scripts\Activate.ps1

```
ursa --help          # verify the command and view options
ursa --print-config  # inspect URSA defaults
```

Optional dashboard: Install “ursa-ai[dashboard]” instead of ursa-ai, then verify with ursa-dashboard --help.

2 · Connect to the Model Access Gateway (MAG)

Step 1 – Get an American Science Cloud (AmSC) Account

Follow the instructions listed here: https://amsc-docs-d762d2.gitlab.io/GM-getting-started#access--login to create an AmSC account. Once the account is approved, proceed to the next step.

Step 2 — Request MAG access and generate a Personal Access Token (PAT)

Follow the instructions listed here: https://amsc-docs-d762d2.gitlab.io/model-access-gateway#quick-start

Step 3 — Put the key in an environment variable

Add the appropriate form to your shell profile, then start a new terminal. The YAML configuration will refer to the variable name rather than containing the secret.

bash / zsh	PowerShell
export AMSC_I2_API_KEY="<your-MAG-key>"	`$env:AMSC_I2_API_KEY="<your-MAG-key>"`

Keep secrets local: Do not commit project keys to config.yaml, source control, prompts, shared agent files, or project context. Use environment variables or an approved secret store.

Step 3 — Find a model ID available to your key

MAG access is project-specific. Query the OpenAI-compatible model list and choose an ID returned for your project; do not assume every key has the same models.

```
curl -s https://i2-api.genesis.american-science-cloud.org/models \
  -H "Authorization: Bearer $AMSC_I2_API_KEY"
```

3 · Create a Reusable URSA Configuration

Create config.yaml beside your project. Replace <MAG-model-id> with an ID returned by /v1/models. The openai: prefix tells URSA to use its OpenAI-compatible provider integration.

```
llm_model:
```

```
  model: openai:<MAG-model-id>
  base_url: https://i2-api.genesis.american-science-cloud.org/
  api_key_env: AMSC_I2_API_KEY

workspace: .
group: default
use_web: false
```

4 · Start URSA and Validate the Connection

Launch URSA from the project directory. URSA creates the configured workspace if it does not already exist.

```
ursa --config config.yaml
```

A successful launch shows the configured LLM endpoint and model, followed by the ursa> prompt. Inside the REPL:

Command	What it checks
help or ?	`List available interactive commands`
models	`Show the active LLM and embedding model`
agents	`Show configured agents and their settings`
exit	`Leave the REPL; Ctrl+D also exits on macOS/Linux`

Connection errors: Confirm the key is present in the current shell, the YAML uses the exact environment-variable name, base_url ends in /v1, and the selected model ID is available to your MAG project key.

5 · Run a First Agentic Workflow

Plain text goes to the default chat behavior. Prefix a request with an agent name to invoke that specialist.

```
ursa> Summarize what URSA can help me do.

ursa> plan Create a test plan for validating analysis.py.

ursa> execute Write and run a Python script that prints the first 10 primes.
```

URSA passes the previous agent's result into the next agent's prompt. This supports a simple plan-then-execute sequence:

```
ursa> plan Build a careful analysis plan for data.csv.
ursa> execute Execute the plan step-by-step and report every generated file.
```

Workspace safety: The execution agent can write files and run shell commands. Use a dedicated workspace or a disposable copy of important data, review actions and generated code, and keep backups of anything you cannot risk modifying.

6 · Useful Launch Patterns

Pattern	Command
Reusable config	`ursa --config config.yaml`
Named persistent agent	`ursa --config config.yaml --name my-agent`
Noninteractive prompt	`ursa --config config.yaml exec "Summarize this project."`
Opt in to web tools	`ursa --config config.yaml --use-web`
Launch dashboard	`ursa-dashboard --host 127.0.0.1 --port 8080`
Launch dashboard with web tools	`URSA_DASHBOARD_USE_WEB=1 ursa-dashboard --host 127.0.0.1 --port 8080`

Web tools are opt in. Enable --use-web only when network requests through supported web/search tools are appropriate for the task and data.

7 · Everyday Commands

Command	What it does
ursa --help	`Show CLI options and subcommands`
ursa --print-config	`Print the resolved configuration and defaults`
ursa --config config.yaml	`Start the interactive URSA REPL`

Command	What it does
help / ?	List commands inside the REPL
models	Display the active model configuration
agents	Display available agents and their settings
plan <task>	Ask the planning agent to develop a plan
execute <task>	Ask the execution agent to work in the configured workspace
exit	Exit the interactive session