Project Proposal Google Summer of Code 2026

Subcommand CLI for GRASS

- Project length: Large (350 hours)
- Rating: medium
- Requirements: Python, Python API, GRASS, GeoTiff, GeoTiff, Technical Writing

---

1. Abstract

1.1 Overview

The current GRASS GIS command-line interface requires a specific environment and multiple steps (data import, region setting, execution, and export) to run a single analysis. This project aims to modernize the GRASS CLI by implementing a subcommand structure, specifically focusing on a new `grass run` command. The goal is to allow users to execute GRASS modules directly on external files like GeoTiff or GeoPackage in a single, streamlined step.

By developing a robust Python API and extending the `grass.app` infrastructure, I will enable high-level features such as:

- **One-Step Execution:** Running tools like `r.slope.aspect` directly on an external `.tiff` file.
- **In-Place Data Handling:** Using `--import-raster/vector` and `--link-raster/vector` flags to manage data without manual pre-processing.
- **Contextual Flexibility:** Adding `--region` support to automatically handle temporary computational regions during execution.

This modernization will lower the barrier to entry for new users and significantly speed up workflows for power users who rely on automated scripting.

1.2 Goals

The primary objective is to transform the GRASS CLI into a modern, user-friendly tool that aligns with contemporary data science workflows. The project will focus on these key areas:

- **Unified Subcommand Structure:** Implement a robust `grass run` command to manage modular executions efficiently.
- **One-Step Data Integration:** Develop `--import-raster/vector` and `--link-raster/vector` flags to eliminate manual pre-processing steps.
- **Dynamic Context Management:** Introduce a `--region` flag to automate the creation and cleanup of temporary computational regions for each command.
- **CLI Parity & Python API:** Ensure 100% parity with existing CLI functionalities through a clean, scalable Python API.

1.3 Benefits

Improved Accessibility: Lowering the barrier for new users who are familiar with modern CLI tools but find the traditional GRASS environment complex.

Efficiency for Power Users: Significant reduction in the number of commands required to perform standard analyses on external files.
Better Integration: Making GRASS tools easily callable from external scripts, CI/CD pipelines, and cloud-native environments.

---

2. Approach

2.1 Environment & Core Infrastructure

- Development Environment: I will establish a robust development environment on macOS (m4) , utilizing `git` for version control and ensuring full compatibility with the GRASS GIS 8.5 core.
  `grass.app` Expansion: The core implementation involves extending the `grass.app` Python package to support a recursive subcommand structure using the `argparse` library, enabling the `grass run [module]` syntax.

2.2 Implementing Core Flags and Logic

- One-Step Execution Logic: I will design a wrapper that handles the entire lifecycle of a command: importing/linking data, setting the region, executing the module, and optionally exporting the result.
- Automated Data Handling: For `--import-raster/vector`, the API will internally call `r.import` or `v.import` to bring external files (GeoTiff/GeoPackage) into a temporary mapset.
- Temporary Region Management: The `--region` flag will utilize `g.region` logic to set a computational extent that exists only for the duration of the command, ensuring no permanent workspace changes.

2.3 Priority Modules for CLI Parity To ensure "complete parity with the existing CLI," I will prioritize the following module groups:

- Raster Analysis: `r.slope.aspect`, `r.mapcalc`, and `r.buffer` will serve as pilot modules for testing the one-step execution workflow.
- Vector Network: Building on my previous experience, I will ensure full support for `v.net` modules (e.g., `v.net.path`, `v.net.alloc`) within the new subcommand structure.
- Metadata & Query: Integration of `g.list`, `g.region`, and `v.info` to provide essential data insights through the CLI.

2.4 Testing & Quality Assurance

- Unit Testing: Using the `gunittest` framework, I will develop automated tests to verify argument parsing and environment cleanup.
- Integration Tests: I will create "End-to-End" scenarios (e.g., running `r.slope.aspect` **on a raw `.tif` file) to validate the "one-step" promise of the project.**

2.5 What have I achieved so far?

- **System Maintenance & Recovery:** I identified a critical build failure in the **Hugo static site generator** infrastructure after cloning the repository; by troubleshooting the version-specific errors in my local environment, I developed a cross-compatible solution for both legacy and modern Hugo versions, which was successfully merged into the official GRASS GIS website.
- **Documentation & Bug Fixing:** While working on the `grass-addons` repository, I discovered inconsistencies in existing examples in r.neighborhoodmatrix I corrected these errors to ensure functional clarity for future developers, and my fix was merged. It was a minor fix.
- **Deep Integration (C & Python):** While working on the "Add JSON Support" project, I gained practical experience with the GRASS GIS core by implementing Python-based tests and developing C code, which allowed me to understand their rigorous coding standards, attention to detail, and the use of standardized parameters like G_OPT_F_FORMAT for generic functionality.
- **Technical Learning:** As a student, it was fascinating to discover how much spatial data is embedded within a single .tif file, and by working with both raster and vector structures, I learned how GRASS manages these datasets within an SQLite database and routes information through its APIs to handle complex map results.
- **Active GSoC Task Progress:** I am currently implementing the `--link-raster` and `--link-vector` requirements (using `r.external` and `v.external`) as specified in the GSoC 2026 Ideas page. I have already opened a Pull Request for these tasks, including `vector.out` and `r.external.out` integrations to visualize results. I am actively refining this PR for final approval.
- **Ongoing Research:** I have recently begun technical research and prototyping for the `--region` support, which is a key component of the Subcommand CLI proposal.

2.6 Code Explanation and Examples:

External Data Linking: I used r.external and v.external with the input and output parameters to link spatial files to the session, using the -o flag to skip projection checks and Path().stem to automate internal map naming.

Raster–external and vector-external in below:

```
    if args.link_raster:
        for raster in args.link_raster:
            gs.run_command(
                "r.external",
                input=raster,
                output=Path(raster).stem,
                flags="o",
                env=session.env,
                errors="status",
            )
            gs.run_command("g.list", type="raster", env=session.env)
    if args.link_vector:
        for vector in args.link_vector:
            gs.run_command(
                "v.external",
                input=vector,
                output=Path(vector).stem,
                flags="o",
                env=session.env,
                errors="status",
            )
```

Raster–external and vector-external run commands in below:

```
create_parser.add_argument(
    "--link-raster",
    action="append",
    help="link a raster map to the project (can be used multiple times)",
)
create_parser.add_argument(
    "--link-vector",
    action="append",
    help="link a vector map to the project (can be used multiple times)",
)
```

External Output Management: I implemented r.external.out with the directory and format="GTiff" parameters to save rasters externally, and v.external.out with the output and format="GPKG" parameters to route new vector layers directly to a GeoPackage.

Raster-out and vector-out in below:

```python
if args.out_raster:
    for out_r in args.out_raster:
        abs_out_path = os.path.abspath(out_r)
        Path(abs_out_path).mkdir(exist_ok=True, parents=True)
        gs.run_command(
            "r.external.out",
            directory=abs_out_path,
            env=session.env,
            errors="status",
            format="GTiff",
        )
if args.out_vector:
    for out_v in args.out_vector:
        abs_out_path = os.path.abspath(out_v)
        Path(abs_out_path).mkdir(exist_ok=True, parents=True)
        gs.run_command(
            "v.external.out",
            output=out_v,
            env=session.env,
            errors="status",
            format="GPKG",
        )      You, 3 days ago • add subcommand for --link-raste
```

Raster-out and vector-out run commands in below:

```python
create_parser.add_argument(
    "--out-raster",
    action="append",
    help="define external format for raster output (can be used multiple times)",
)
create_parser.add_argument(
    "--out-vector",
    action="append",
    help="define external format for vector output (can be used multiple times)",
)
```

How to Test them:

I tested the raster linking functionality using the `ortho2010_t792_subset_20cm.tif` file from the `nc_spm_08_grass7`dataset, ensuring the map name matched the filename. For vector testing, I used the ne_110m_admin_0_countries dataset from Natural Earth as an external sample to verify the link process.
https://www.naturalearthdata.com/downloads/110m-cultural-vectors/110m-admin-0-countries/.
To run my commands successfully you need to ensure that these files are in your workplaces.

Before starting the test, ensure that you have copied your updated cli.py to the GRASS installation directory.

cp cli.py /usr/local/grass85/etc/python/grass/app

To Test Image File

python3 -m grass.app run r.info --link-raster=ortho2010_t792_subset_20cm.tif map=ortho2010_t792_subset_20cm.1

Result

| Type of Map: GDAL-link Number of Categories: 0 |
| Data Type: CELL Semantic label: (none) |
| Rows: 3750 |
| Columns: 3500 |
| Total Cells: 13125000 |
| Projection: x,y |
| N: 220750 S: 220000 Res: 0.2 |
| E: 639000 W: 638300 Res: 0.2 |
| Range of data: min = 0 max = 251 |
| |
| Data Source: |
| /Users/tangel/Works/grass/ortho2010_t792_subset_20cm.tif band 1 |
| |
| |
| Data Description: |
| generated by r.external |
| |
| Comments: |
| r.external -o input="ortho2010_t792_subset_20cm.tif" output="ortho20\ |
| 10_t792_subset_20cm"

To Test Map file

python3 -m grass.app run v.info --link-vector=/ne_110m_admin_0_countries/ne_110m_admin_0_countries.shp map=ne_110m_admin_0_countries

Result

OGR layer: ne_110m_admin_0_countries | 18:01 [260/653]
| OGR datasource: /Users/tangel/Downloads/ne_110m_admin_0_countries/ne_110m |

Feature type: polygon
Type of map: vector (level: 2)
Number of points: 0 Number of centroids: 288
Number of lines: 0 Number of boundaries: 289
Number of areas: 289 Number of islands: 289
Map is 3D: No
Number of dblinks: 1
Projection: x,y
N: 83.64513 S: -90
E: 180 W: -180
Digitization threshold: 0
Comment:

To Test Raster.out

python3 -m grass.app run r.mapcalc "test_result = ortho2010_t792_subset_20cm.1" --link-raster=ortho2010_t792_subset_20cm.tif --out-raster=./test_salih

Result

Over-riding projection check Reading band 1 of 3…

Link to raster map <ortho2010_t792_subset_20cm.1> created.
Reading band 2 of 3... Link to raster map <ortho2010_t792_subset_20cm.2> created.
Reading band 3 of 3... Link to raster map <ortho2010_t792_subset_20cm.3> created.
Imagery group <ortho2010_t792_subset_20cm> created
ortho2010_t792_subset_20cm.1 ortho2010_t792_subset_20cm.2
ortho2010_t792_subset_20cm.3
100%

To Test Vector.out

python3 -m grass.app run v.category input=ne_110m_admin_0_countries option=report 19:20
[5/1404]
t \
--link-vector=./ne_110m_admin_0_countries/ne_110m_admin_0_countries.shp \
--out-vector=./test_salih/test_result.gpkg

Result

Over-riding projection check 19:20 [1/1404]
Building topology for vector map ne_110m_admin_0_countries@PERMANENT...
Using external data format 'ESRI Shapefile' (feature type 'polygon')
Registering primitives...
577 primitives registered
10654 vertices registered
v.external complete. Link to vector map <ne_110m_admin_0_countries>
created.
Current output format for vectors: GPKG
Layer/table: 1/ne_110m_admin_0_countries
type count min max
point 0 0 0
line 0 0 0
boundary 289 0 176
centroid 0 0 0
area 0 0 0
face 0 0 0
kernel 0 0 0
all 289 0 176

2.7 What can be done more for this project?

Which subcommands or sub-subcommands can be implemented:

- **Region Management:** A new `--region` flag can be added to automatically set the computational boundaries (using `g.region`) before running a module.
- **Metadata Inspection:** A subcommand like `info` to quickly display file properties (using `r.info` or `v.info`) without needing to link the data first.

What the Grass CLI needs more now and in the future:

- **Improved Error Handling:** Providing more user-friendly messages when a file path is wrong or a format is unsupported.
- **Format Expansion:** Adding support for more common file types in the `--out-raster` and `--out-vector` flags.
- **Mentor Guidance:** I plan to consult with my mentors to prioritize which additional GRASS modules should be integrated into the CLI for a more complete workflow.

---

3. Timeline & Commitment

Continuous Commitment: I am fully aware that consistent communication is the backbone of a successful GSoC project. I formally commit to providing **weekly progress reports every Monday** (or daily updates via mailing lists/Slack as preferred by the mentors) to ensure transparency and timely feedback throughout the entire duration of the program.

3.1 Phase 1: Community Bonding & Architectural Design (May 8 - June 1)

- **Week 1-2:** Deep dive into the `grass.app` source code and existing experimental CLI logic. Identify architectural constraints and establish a robust local development/testing environment.
- **Week 3-4: Collaborative Planning:** Work closely with mentors to finalize the selection of subcommands and options. Instead of hardcoding a fixed list, I will focus on defining the *priority criteria* for which modules should achieve parity first, ensuring the project aligns with the community's immediate needs.

3.2 Phase 2: Core Implementation & Subcommand Infrastructure (June 2 - July 20)

- **Week 5-7:** Implementation of the base `grass run` infrastructure and the underlying Python API. This phase focuses on the "engine" that will handle subcommand routing and argument parsing.
- **Week 8-10: Dynamic Logic Integration:** Development of core functionalities like automated data linking/importing and temporary region management. The specific

subcommands to be implemented will be finalized here based on the Phase 1 discussions.
- **Week 11-12:** Testing the integration of the selected subcommands and ensuring they handle external files (GeoTiff/GeoPackage) seamlessly in a "one-step" workflow.

3.3 Phase 3: Parity, Refinement & Edge Cases (July 21 - August 15)

- **Week 13-14:** Expanding the subcommand coverage to achieve the agreed-upon parity. Refinement of the user interface based on initial feedback from mentors and the OSGeo community.
- **Week 15:** Intensive testing across different datasets and operating systems to ensure the new CLI is robust and fails gracefully when encountering invalid inputs.

3.4 Phase 4: Final Documentation & Reporting (August 16 - September 1)

- **Week 16: Final Reporting & Documentation:** Comprehensive update of the GRASS manual pages and wiki. Writing the final project report, summarizing the achievements, and outlining future maintenance steps. This period is dedicated to ensuring the code is "merge-ready" and well-documented for the long term.

4.Relevant Contributions

The following contributions are included to demonstrate my familiarity with the GRASS GIS ecosystem, its codebase standards, and the organizational workflow.

Discourse Topics (With Mentors) :
https://discourse.osgeo.org/t/gsoc-2026-adding-json-output-support-to-v-build/152360
https://discourse.osgeo.org/t/gsoc-2026-add-json-output-to-different-tools-in-c/151822

Opened Issues:
https://github.com/OSGeo/grass/issues/6942
https://github.com/OSGeo/grass-website/issues/604
https://github.com/OSGeo/grass-addons/issues/1637
https://github.com/OSGeo/grass/issues/7176(Related)

Merged Branches:
https://github.com/OSGeo/grass-website/pull/605
https://github.com/OSGeo/grass-addons/issues/1637

Pull Requests:
https://github.com/OSGeo/grass/pull/7182(Most Related)
https://github.com/OSGeo/grass/pull/7088

---

5. About Me

5.1 Contact Information

Personal Information:

- **Name:** Salih Tangel
- **Email:** salihtangel@gmail.com
- **University:** Kocaeli University, Kocaeli, Turkey
- **Github**: https://github.com/tangelll
- **TimeZone**: (UTC +3:00)
- **Linkedin:https://linkedin.com/in/salih-tangel**

5.2 Post Gsoc

Long-term Commitment: I recognize that open-source projects thrive on long-term maintainership. I commit to staying active in the OSGeo community, maintaining the modules I develop, and helping new contributors after GSoC concludes.

5.3 References

I used as an example the Open-source Gemini Example Apps" proposal for Google Summer of Code 2025 by Triyan Mukherjee.