MCP Server Tool Ideas

This document lists potential new tools to enhance the capabilities of the MCP server, focusing on terminal commands, file system interactions, and process management.

Existing Tools

1. `runShellCommand`: Execute arbitrary shell commands.
2. `runPythonFile`: Execute a specific Python script.
3. `readDirectory`: List contents of a directory.
4. `copyFile`: Copy a file.
5. `createFile`: Create a new file with content.

Proposed New Tools and Integration Analysis

Context-Aware Operations

0. **`combinationTask`**: ☑ INTEGRABLE
    - Description: Contains two input fields, one to select directory and the other to execute a task within that directory
    - Integration: Can be implemented as a higher-level tool that uses the workspace path resolution already in place
    - Implementation: Create a wrapper function that sets the working directory before executing the command
    - MCP Compatibility: Fully compatible with MCP server architecture

Basic File Operations

1. **`editFile`**: ☑ INTEGRABLE

    - Description: Modify the content of an existing file (append text, replace lines, insert at specific points)
    - Integration: Can use Node.js fs module with the existing workspace path resolution
    - Implementation: Create a function that reads a file, modifies its content, and writes it back
    - MCP Compatibility: Fully compatible with MCP server architecture

2. **`readFile`**: ☑ INTEGRABLE

    - Description: Read the entire content or specific lines of a file
    - Integration: Can use Node.js fs.readFile with the existing workspace path resolution
    - Implementation: Create a function that reads a file and returns its content
    - MCP Compatibility: Fully compatible with MCP server architecture

3. **`deleteFile`**: ☑ INTEGRABLE

    - Description: Remove a specified file
    - Integration: Can use Node.js fs.unlink with the existing workspace path resolution
    - Implementation: Create a function that deletes a file with proper error handling

- MCP Compatibility: Fully compatible with MCP server architecture

4. `moveFile`: ☑ INTEGRABLE
   - Description: Move a file from one location to another
   - Integration: Can use Node.js fs.rename with the existing workspace path resolution
   - Implementation: Create a function that moves a file with proper error handling
   - MCP Compatibility: Fully compatible with MCP server architecture

Directory Operations

5. `createDirectory`: ☑ INTEGRABLE
   - Description: Create a new directory
   - Integration: Can use Node.js fs.mkdir with the existing workspace path resolution
   - Implementation: Create a function that creates a directory with proper error handling
   - MCP Compatibility: Fully compatible with MCP server architecture

6. `moveDirectory`: ☑ INTEGRABLE
   - Description: Move a directory from one location to another
   - Integration: Can use Node.js fs.rename with the existing workspace path resolution
   - Implementation: Create a function that moves a directory with proper error handling
   - MCP Compatibility: Fully compatible with MCP server architecture

7. `copyDirectory`: ☑ INTEGRABLE
   - Description: Copy a directory and its contents to another location
   - Integration: Requires a recursive function using fs.mkdir and fs.copyFile
   - Implementation: Create a recursive function that copies directories and files
   - MCP Compatibility: Fully compatible but requires careful implementation for large directories

8. `getDirectoryTree`: ☑ INTEGRABLE
   - Description: Create a tree view of a directory and its contents
   - Integration: Can build on the existing readDirectory function with recursion
   - Implementation: Create a recursive function that builds a tree structure
   - MCP Compatibility: Fully compatible but may need pagination for large directories

File Search and Filter Operations

9. `grep`: ☑ INTEGRABLE
   - Description: Search for specific patterns in files
   - Integration: Can use Node.js readline interface to search files line by line
   - Implementation: Create a function that searches files for patterns
   - MCP Compatibility: Fully compatible but may need optimization for large files

File Validation and Comparison

10. `compareFiles`: ☑ INTEGRABLE
    - Description: Compare two files for differences (similar to diff)

- Integration: Can use Node.js fs.readFile and implement a diff algorithm
- Implementation: Create a function that compares files and returns differences
- MCP Compatibility: Fully compatible but may need optimization for large files

Process Information

11. `listProcesses`: ☑ INTEGRABLE
    - Description: List currently running processes
    - Integration: Can use a third-party library like ps-list or native commands
    - Implementation: Create a function that lists processes with proper formatting
    - MCP Compatibility: Fully compatible but requires platform-specific handling
12. `getProcessInfo`: ☑ INTEGRABLE
    - Description: Get detailed information about a specific process
    - Integration: Can use a third-party library or native commands
    - Implementation: Create a function that gets process details
    - MCP Compatibility: Fully compatible but requires platform-specific handling
13. `getProcessTree`: ☑ INTEGRABLE
    - Description: Display the process hierarchy as a tree
    - Integration: Can build on listProcesses with parent-child relationship mapping
    - Implementation: Create a function that builds a process tree
    - MCP Compatibility: Fully compatible but requires platform-specific handling
14. `findProcessByName`: ☑ INTEGRABLE
    - Description: Find processes with a specific name
    - Integration: Can build on listProcesses with filtering
    - Implementation: Create a function that filters processes by name
    - MCP Compatibility: Fully compatible but requires platform-specific handling
15. `findProcessByUser`: ☑ INTEGRABLE
    - Description: Find processes belonging to a specific user
    - Integration: Can build on listProcesses with filtering
    - Implementation: Create a function that filters processes by user
    - MCP Compatibility: Fully compatible but requires platform-specific handling
16. `getProcessCommandLine`: ☑ INTEGRABLE
    - Description: Get the command line used to start a process
    - Integration: Can use a third-party library or native commands
    - Implementation: Create a function that gets process command line
    - MCP Compatibility: Fully compatible but requires platform-specific handling

Process Control

17. `killProcess`: ☑ INTEGRABLE

- Description: Terminate a process by its ID or name
- Integration: Can use Node.js process.kill or native commands
- Implementation: Create a function that kills a process with proper error handling
- MCP Compatibility: Compatible with basic safety checks

Background Jobs and Services

19. `checkJobStatus`: ☑ INTEGRABLE
    - Description: Check the status of a background job
    - Integration: Requires job tracking system implementation
    - Implementation: Create a job tracking system and status checking function
    - MCP Compatibility: Compatible but requires additional infrastructure
20. `listBackgroundJobs`: ☑ INTEGRABLE
    - Description: List running background jobs
    - Integration: Requires job tracking system implementation
    - Implementation: Create a job tracking system and listing function
    - MCP Compatibility: Compatible but requires additional infrastructure
21. `listSystemServices`: ☑ INTEGRABLE
    - Description: List available system services
    - Integration: Can use platform-specific commands
    - Implementation: Create platform-specific functions for service listing
    - MCP Compatibility: Compatible but platform-dependent
22. `getServiceLogs`: ☑ INTEGRABLE
    - Description: Retrieve logs for a system service
    - Integration: Can use platform-specific commands
    - Implementation: Create platform-specific functions for log retrieval
    - MCP Compatibility: Compatible but platform-dependent

Network and Connection Management

23. `portChecking`: ☑ INTEGRABLE
    - Description: Check if ports are in use on the system
    - Integration: Can use Node.js net module
    - Implementation: Create functions for port checking and listing
    - MCP Compatibility: Fully compatible with MCP server architecture

Log Management

26. `readLogFile`: ☑ INTEGRABLE
    - Description: Read and display the contents of a log file
    - Integration: Can use Node.js fs.readFile with the existing workspace path resolution
    - Implementation: Create a function that reads log files with proper formatting
    - MCP Compatibility: Fully compatible with MCP server architecture

27. `findInLogs`: ☑ INTEGRABLE
    - Description: Search for specific patterns in log files
    - Integration: Can build on readLogFile with pattern matching
    - Implementation: Create a function that searches log files for patterns
    - MCP Compatibility: Fully compatible but may need optimization for large files

Integration Summary

- **Fully Integrable (Low Complexity)**: 22 tools
- **Not Implementing Now**: 0 tools

All proposed tools can be integrated into the MCP server architecture with reasonable effort. The implementation should follow a phased approach, starting with the file operations and progressing to more specialized tools.