GitHub Copilot Agents 🤖

GitHub Copilot Agents are specialized AI assistants that extend beyond the standard chat participants. They provide domain-specific expertise, custom workflows, and enhanced capabilities for specific development scenarios.

🎯 What Are Copilot Agents?

Agents are intelligent assistants that can:

- **Understand Context**: Analyze your entire project structure and dependencies
- **Execute Tasks**: Perform complex multi-step operations automatically
- **Learn Patterns**: Adapt to your coding style and project conventions
- **Integrate Tools**: Connect with external services and development tools

Built-in vs Custom Agents

Built-in Agents	Custom Agents
@workspace, @github, @vscode, @terminal	Organization-specific agents
Pre-trained on common tasks	Trained on internal codebases
Available to all users	Enterprise/team exclusive
Integrated with model picker	Custom model selection

🚀 Getting Started with Agents

Basic Agent Interaction

```
# In Copilot Chat
@github help me create a GitHub Actions workflow for Node.js

@vscode show me keyboard shortcuts for debugging

@terminal help me set up environment variables
```

Advanced Agent Usage

```
# Multi-step workflow with context
@workspace analyze the architecture of this Spring Boot app and suggest
improvements for scalability

# Agent chaining
@github create a PR template, then @vscode configure the editor settings
for this project
```

🛠️ Available Built-in Agents

@github Agent

Specializes in: Repository management, CI/CD, collaboration

```
# Examples
@github create a comprehensive .gitignore for Java Spring Boot
@github suggest branch protection rules for this repository
@github help me write a good PR description for this feature
```

Best Practices: ✅ **Do**: Be specific about your repository context ✅ **Do**: Mention specific GitHub features you need ❌ **Don't**: Ask about non-GitHub related tasks

@workspace Agent

Specializes in: Project analysis, architecture, cross-file understanding

```
# Examples
@workspace find all TODO comments across the codebase
@workspace explain the data flow in this application
@workspace suggest refactoring opportunities for better maintainability
```

Best Practices: ✅ **Do**: Use for project-wide analysis ✅ **Do**: Ask about architectural patterns ❌ **Don't**: Use for single-file questions

@vscode Agent

Specializes in: Editor configuration, productivity, debugging

```
# Examples
@vscode configure launch.json for debugging this Spring Boot app
@vscode suggest extensions for Java development
@vscode help me optimize my workspace settings
```

Best Practices: ✅ **Do**: Ask about IDE-specific features ✅ **Do**: Request configuration help ❌ **Don't**: Use for general coding questions

@terminal Agent

Specializes in: Command-line operations, scripts, system administration

```
# Examples
@terminal create a script to build and deploy this application
@terminal help me debug this Maven build failure
@terminal suggest environment setup commands
```

Best Practices: ✅ **Do**: Specify your operating system ✅ **Do**: Include error messages ❌ **Don't**: Ask for GUI-related help

🏢 Enterprise Agents

Custom Organization Agents

Organizations can create specialized agents for:

- **Internal APIs**: Documentation and usage guidance
- **Coding Standards**: Automated code review and suggestions
- **Deployment**: Custom CI/CD pipeline management
- **Security**: Vulnerability scanning and compliance

Agent Development

```
// .github/copilot/agents/internal-api.json
{
  "name": "internal-api",
  "description": "Expert on company's internal API standards",
  "knowledge_base": [
    "docs/api-guidelines.md",
    "examples/api-patterns/**"
```

```
  ],
  "tools": [
    "api-validator",
    "schema-generator"
  ]
}
```

💡 Advanced Agent Techniques

Agent Combinations

```
# Sequential agent workflow
@workspace identify all database queries, then @github create an issue to
optimize them

# Parallel agent consultation
@terminal && @vscode help me set up a development environment with
debugging capabilities
```

Context-Rich Requests

```
# Include specific context
@github #file:package.json #file:README.md suggest improvements to this
project's documentation and dependencies

# Reference specific patterns
@workspace #selection analyze this code pattern and find similar
implementations elsewhere
```

Agent Specialization

```
# Domain-specific requests
@github help me set up semantic versioning for this library
@vscode configure ESLint rules that match our team's coding standards
@terminal create deployment scripts for AWS ECS
```

🎨 Best Practices

✅ Do's

1. **Be Specific**: Mention exact technologies, frameworks, and versions

```markdown # Good @workspace analyze this Spring Boot 3.2.3 application using Java 21

# Bad @workspace analyze this Java app ```

1. **Provide Context**: Include relevant files and current state

`markdown @github #file:pom.xml #file:src/main/resources/application.properties help me configure CI/CD for this Spring Boot app`

1. **Use Agent Strengths**: Match the request to the agent's specialization

`markdown @terminal help me with Maven commands @vscode help me with IDE configuration @github help me with repository settings`

1. **Chain Agents Logically**: Sequence agents for complex workflows

`markdown @workspace identify the main components, then @github create issues for each component that needs testing`

## ❌ Don'ts

1. **Don't Mix Domains**: Keep agent requests focused

```markdown # Bad @github help me fix this Java syntax error and configure my IDE

# Good @vscode help me configure my IDE for Java development ```

1. **Don't Ignore Agent Scope**: Use agents within their expertise

```markdown # Bad @terminal help me design a database schema

# Good @workspace help me design a database schema for this application ```

1. **Don't Overload Requests**: Keep individual requests focused

```markdown # Bad @workspace analyze everything and fix all issues and create documentation

# Good @workspace analyze the current architecture and identify improvement opportunities ```

# 🚨 Common Pitfalls

## Agent Confusion

**Problem**: Using wrong agent for task **Solution**: Check agent specialization before asking

## Context Overload

**Problem**: Providing too much irrelevant context **Solution**: Focus on context relevant to the specific request

## Expectation Mismatch

**Problem**: Expecting agents to perform actions they can't **Solution**: Understand each agent's capabilities and limitations

# 🔧 Troubleshooting

## Agent Not Responding

1. Check agent availability in your organization
2. Verify proper syntax (use `@` prefix)
3. Ensure context is accessible to the agent

## Poor Agent Responses

1. Add more specific context
2. Use the correct agent for the task
3. Break complex requests into smaller parts

## Agent Limitations

1. Agents work within VS Code's security constraints
2. External tool access depends on organization policies
3. Some features require enterprise licenses

# 🎓 Learning Exercises

## Beginner

1. Try each built-in agent with a simple request
2. Compare responses from different agents for the same question
3. Practice providing context with `#file` and `#selection`

## Intermediate

1. Chain multiple agents for a complete workflow
2. Use agents to analyze and improve an existing project
3. Create custom prompts that leverage agent specializations

## Advanced

1. Design agent workflows for complex development tasks
2. Evaluate agent responses for accuracy and usefulness
3. Contribute to organization-specific agent development

---

**Remember**: Agents are tools to enhance your development workflow. The key to success is understanding each agent's strengths and using them appropriately for your specific needs.