Cómo Agregar Nuevas Herramientas al Agente

Este documento explica cómo crear nuevas herramientas (tools) para extender las capacidades del agente personal.

Arquitectura del Sistema de Herramientas

El agente usa **Function Calling** para decidir automáticamente qué herramienta usar según la solicitud del usuario:

```
Usuario: "Agenda reunión mañana a las 3pm"
    ↓
Agente (LLM) → Decide usar: calendar_create_event
    ↓
Orquestador → Ejecuta la herramienta
    ↓
Agente (LLM) → Recibe resultado y responde al usuario
```

Componentes

1. **Tool (clase base)** - Define la interfaz de una herramienta
2. **ToolRegistry** - Registro central de todas las herramientas
3. **PersonalAgent** - Orquestador que coordina las llamadas

Paso a Paso: Crear una Nueva Herramienta

1. Crear el archivo de la herramienta

Crea un nuevo archivo en `src/tools/` (ej: `email_tool.py`):

```python
"""Herramienta para gestión de emails."""
import logging
from typing import Dict, Any, List
from .base import Tool, ToolParameter

logger = logging.getLogger(__name__)


class EmailSendTool(Tool):
    """Herramienta para enviar emails."""

    @property
    def name(self) -> str:
        # Nombre único que el LLM usará para llamar a esta herramienta
        return "email_send"

    @property
```

```
    def description(self) -> str:
        # Descripción que ayuda al LLM a decidir cuándo usar esta
herramienta
        return (
            "Envía un email al destinatario especificado. "
            "Úsala cuando el usuario quiera enviar un correo electrónico."
        )

    @property
    def parameters(self) -> List[ToolParameter]:
        # Define los parámetros que acepta esta herramienta
        return [
            ToolParameter(
                name="to",
                type="string",
                description="Dirección de email del destinatario",
                required=True
            ),
            ToolParameter(
                name="subject",
                type="string",
                description="Asunto del email",
                required=True
            ),
            ToolParameter(
                name="body",
                type="string",
                description="Cuerpo del mensaje",
                required=True
            ),
            ToolParameter(
                name="cc",
                type="array",
                description="Lista de emails en copia (opcional)",
                required=False
            )
        ]

    async def execute(self, **kwargs) -> Dict[str, Any]:
        """
        Ejecuta el envío del email.

        Args:
            to: Destinatario
            subject: Asunto
            body: Cuerpo del mensaje
            cc: Lista de CCs (opcional)

        Returns:
            Dict con el resultado de la operación
        """
        to = kwargs.get("to")
        subject = kwargs.get("subject")
        body = kwargs.get("body")
```

```
        cc = kwargs.get("cc", [])

        try:
            # AQUÍ VA TU INTEGRACIÓN REAL
            # Por ejemplo:
            # import smtplib
            # smtp.sendmail(from, to, message)

            logger.info(f"Email enviado a: {to} - Asunto: {subject}")

            return {
                "success": True,
                "message": f"Email enviado exitosamente a {to}",
                "email_id": "msg_123456"  # ID del email enviado
            }

        except Exception as e:
            logger.error(f"Error enviando email: {e}")
            return {
                "success": False,
                "error": f"Error enviando email: {str(e)}"
            }
```

2. Registrar la herramienta en `__init__.py`

Edita `src/tools/__init__.py`:

```
"""Sistema de herramientas para el agente personal."""
from .base import Tool, ToolRegistry
from .calendar_tool import CalendarTool, CalendarGetAgendaTool
from .task_tool import TaskCreateTool, TaskListTool, TaskCompleteTool
from .email_tool import EmailSendTool  # ← Agregar import

__all__ = [
    "Tool",
    "ToolRegistry",
    "CalendarTool",
    "CalendarGetAgendaTool",
    "TaskCreateTool",
    "TaskListTool",
    "TaskCompleteTool",
    "EmailSendTool",  # ← Agregar export
]
```

3. Registrar en el agente

Edita `src/core/agent.py`:

```
# En los imports
from ..tools import (
    ToolRegistry,
    CalendarTool,
    CalendarGetAgendaTool,
    TaskCreateTool,
    TaskListTool,
    TaskCompleteTool,
    EmailSendTool,  # ← Agregar
)

# En el método _register_tools
def _register_tools(self):
    """Registra todas las herramientas disponibles."""
    # Herramientas de calendario
    self.tool_registry.register(CalendarTool())
    self.tool_registry.register(CalendarGetAgendaTool())

    # Herramientas de tareas
    self.tool_registry.register(TaskCreateTool())
    self.tool_registry.register(TaskListTool())
    self.tool_registry.register(TaskCompleteTool())

    # Herramientas de email
    self.tool_registry.register(EmailSendTool())  # ← Agregar

    logger.info(f"{len(self.tool_registry.get_all())} herramientas
registradas")
```

4. ¡Listo! Probar

Ahora el agente puede usar tu herramienta automáticamente:

```
response = await agent.process("Envía un email a juan@example.com con
asunto 'Hola'")
# El agente detectará que debe usar email_send y ejecutará la herramienta
```

Tipos de Parámetros

Los tipos disponibles para `ToolParameter.type`:

- `"string"` - Texto
- `"number"` - Número (int o float)
- `"boolean"` - Verdadero/falso
- `"object"` - Objeto JSON
- `"array"` - Lista/array

Enumeraciones

Para limitar valores a opciones específicas:

```
ToolParameter(
    name="priority",
    type="string",
    description="Prioridad de la tarea",
    required=False,
    enum=["urgent", "high", "medium", "low"]  # Solo estos valores
)
```

Mejores Prácticas

1. Descripciones claras

La `description` es CRUCIAL - debe ayudar al LLM a entender:

- Qué hace la herramienta
- Cuándo usarla
- Qué tipo de solicitudes del usuario mapean a esta herramienta

✅ Bueno:

```
description = (
    "Envía un email al destinatario especificado. "
    "Úsala cuando el usuario quiera enviar un correo, "
    "escribir a alguien, o contactar por email."
)
```

❌ Malo:

```
description = "Envía email"  # Muy vago
```

2. Manejo de errores robusto

Siempre captura excepciones y retorna un formato consistente:

```
try:
    # Tu lógica aquí
    return {
        "success": True,
        "message": "Operación exitosa",
        "data": {...}
    }
except Exception as e:
    logger.error(f"Error: {e}")
    return {
```

```
            "success": False,
            "error": str(e)
        }
```

3. Logging apropiado

Usa el logger para debugging:

```
logger.info(f"Ejecutando {self.name} con {kwargs}")
logger.error(f"Error en {self.name}: {e}")
```

4. Validación de parámetros

Valida los datos antes de usarlos:

```
to = kwargs.get("to")
if not to:
    return {
        "success": False,
        "error": "El parámetro 'to' es requerido"
    }

if "@" not in to:
    return {
        "success": False,
        "error": "Email inválido"
    }
```

Herramientas Compuestas

Puedes crear múltiples herramientas relacionadas:

```
# email_tool.py
class EmailSendTool(Tool):
    """Enviar emails"""
    pass

class EmailReadTool(Tool):
    """Leer emails de la bandeja de entrada"""
    pass

class EmailSearchTool(Tool):
    """Buscar emails por criterio"""
    pass
```

El LLM decidirá cuál usar según el contexto.

Integraciones con Servicios Externos

Ejemplo: GitHub

```
class GitHubCreateIssueTool(Tool):
    @property
    def name(self) -> str:
        return "github_create_issue"

    async def execute(self, **kwargs) -> Dict[str, Any]:
        repo = kwargs.get("repo")
        title = kwargs.get("title")
        body = kwargs.get("body")

        # Usar librería de GitHub
        from github import Github
        g = Github(os.getenv("GITHUB_TOKEN"))
        repo_obj = g.get_repo(repo)
        issue = repo_obj.create_issue(title=title, body=body)

        return {
            "success": True,
            "issue_url": issue.html_url,
            "issue_number": issue.number
        }
```

Testing

Crea tests para tus herramientas:

```
# tests/test_email_tool.py
import pytest
from src.tools.email_tool import EmailSendTool

@pytest.mark.asyncio
async def test_email_send():
    tool = EmailSendTool()

    result = await tool.execute(
        to="test@example.com",
        subject="Test",
        body="Test body"
    )

    assert result["success"] == True
    assert "email_id" in result
```

Debugging

Para ver qué herramientas está llamando el agente:

1. Activa logging en DEBUG:

```
export LOG_LEVEL=DEBUG
```

2. El agente logueará:

```
INFO - Ejecutando 2 herramientas
INFO - Ejecutando herramienta: email_send con args: {'to':
'juan@example.com', ...}
```

Ejemplos de Herramientas Útiles

Ideas para expandir el agente:

- **Navegador**: Abrir URLs, buscar en Google
- **Sistema**: Ejecutar comandos, lanzar aplicaciones
- **Logseq**: Crear notas, buscar en knowledge base
- **Spotify**: Reproducir música, crear playlists
- **Weather**: Consultar clima
- **News**: Obtener noticias recientes
- **Translation**: Traducir texto
- **File Operations**: Leer, escribir, buscar archivos

Recursos

- Documentación OpenAI Function Calling
- Código de referencia: `src/tools/calendar_tool.py`, `src/tools/task_tool.py`
- Logs del agente: `data/logs/agent.log`

¿Preguntas? Revisa el código de las herramientas existentes o consulta los logs para debugging.