Guía de Integraciones Específicas

Esta guía te explica paso a paso cómo integrar aplicaciones específicas con el agente personal.

Tabla de Contenidos

Calcurse

Calcurse es una aplicación de calendario en terminal. Puedes integrarla de **dos formas**:

Opción A: Comandos de Shell (Más simple)

Calcurse tiene una CLI excelente. Puedes llamar comandos directamente:

```
# src/tools/calcurse_tool.py
import subprocess
from datetime import datetime
from typing import Dict, Any, List
from .base import Tool, ToolParameter

class CalcurseCreateEventTool(Tool):
    @property
    def name(self) -> str:
        return "calcurse_create_event"

    @property
    def description(self) -> str:
        return "Crea un evento en Calcurse (calendario local)"

    @property
    def parameters(self) -> List[ToolParameter]:
        return [
            ToolParameter(
                name="title",
                type="string",
                description="Título del evento",
                required=True
            ),
            ToolParameter(
                name="start_time",
                type="string",
```

```
                description="Hora de inicio (formato: YYYY-MM-DD HH:MM)",
                required=True
            ),
            ToolParameter(
                name="duration_minutes",
                type="number",
                description="Duración en minutos",
                required=False
            )
        ]

    async def execute(self, **kwargs) -> Dict[str, Any]:
        title = kwargs.get("title")
        start_time_str = kwargs.get("start_time")
        duration = kwargs.get("duration_minutes", 60)

        try:
            # Parsear fecha
            start_dt = datetime.fromisoformat(start_time_str)

            # Calcular hora de fin
            from datetime import timedelta
            end_dt = start_dt + timedelta(minutes=duration)

            # Formato que entiende calcurse: MM/DD/YYYY @ HH:MM ->
MM/DD/YYYY @ HH:MM
            date_str = start_dt.strftime("%m/%d/%Y")
            start_time = start_dt.strftime("%H:%M")
            end_time = end_dt.strftime("%H:%M")

            # Comando calcurse
            # -a: add appointment
            event_str = f"{date_str} @ {start_time} -> {date_str} @
{end_time} | {title}"

            result = subprocess.run(
                ["calcurse", "-a", event_str],
                capture_output=True,
                text=True,
                check=True
            )

            return {
                "success": True,
                "message": f"Evento '{title}' creado en Calcurse",
                "start": start_time_str,
                "duration": duration
            }

        except subprocess.CalledProcessError as e:
            return {
                "success": False,
                "error": f"Error ejecutando calcurse: {e.stderr}"
            }
```

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

Comandos útiles de Calcurse:

```
# Ver ayuda
calcurse --help

# Agregar evento (appointment)
calcurse -a "12/31/2024 @ 14:00 -> 12/31/2024 @ 16:00 | Reunión importante"

# Agregar tarea (todo)
calcurse -t 3  # prioridad 3
# Luego escribe el título

# Listar eventos de un rango de días
calcurse -r7  # próximos 7 días

# Formato de salida más parseable
calcurse --format-apt="%s|%S|%E|%m"
# %s = start time, %S = start date, %E = end date, %m = message
```

Opción B: Leer/Escribir archivos directamente (Más control)

Calcurse guarda sus datos en archivos de texto en `~/.local/share/calcurse/`:

```
# src/tools/calcurse_tool.py
import os
from pathlib import Path
from datetime import datetime

class CalcurseFileTool(Tool):
    def __init__(self):
        # Ruta del directorio de calcurse
        self.calcurse_dir = Path.home() / ".local" / "share" / "calcurse"
        self.apts_file = self.calcurse_dir / "apts"  # Eventos
        self.todo_file = self.calcurse_dir / "todo"  # Tareas

    async def execute(self, **kwargs) -> Dict[str, Any]:
        title = kwargs.get("title")
        start_time_str = kwargs.get("start_time")
        duration = kwargs.get("duration_minutes", 60)

        try:
            start_dt = datetime.fromisoformat(start_time_str)
```

```
            # Formato de calcurse en el archivo:
            # Eventos: [MM/DD/YYYY @ HH:MM -> MM/DD/YYYY @ HH:MM] Título
            date_str = start_dt.strftime("%m/%d/%Y")
            start_time = start_dt.strftime("%H:%M")

            from datetime import timedelta
            end_dt = start_dt + timedelta(minutes=duration)
            end_time = end_dt.strftime("%H:%M")

            event_line = f"[{date_str} @ {start_time} -> {date_str} @
{end_time}] {title}\n"

            # Escribir al archivo
            with open(self.apts_file, "a") as f:
                f.write(event_line)

            return {
                "success": True,
                "message": f"Evento '{title}' creado en Calcurse"
            }

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

Estructura de archivos de Calcurse:

```
~/.local/share/calcurse/
├── apts        # Eventos (appointments)
├── todo        # Tareas
├── conf        # Configuración
└── notes/      # Notas asociadas a eventos
```

Formato del archivo `apts`:

```
[12/31/2024 @ 14:00 -> 12/31/2024 @ 16:00] Reunión con equipo
[01/15/2025 @ 09:00 -> 01/15/2025 @ 10:30] Dentista
```

Khal

Khal es un calendario basado en vCalendar/iCal. Es más estándar que Calcurse.

Usando comandos de Khal

```python
# src/tools/khal_tool.py
import subprocess
from datetime import datetime
from typing import Dict, Any, List
from .base import Tool, ToolParameter

class KhalCreateEventTool(Tool):
    @property
    def name(self) -> str:
        return "khal_create_event"

    @property
    def description(self) -> str:
        return "Crea un evento en Khal (calendario basado en iCal)"

    @property
    def parameters(self) -> List[ToolParameter]:
        return [
            ToolParameter(
                name="title",
                type="string",
                description="Título del evento",
                required=True
            ),
            ToolParameter(
                name="start_time",
                type="string",
                description="Hora de inicio ISO format",
                required=True
            ),
            ToolParameter(
                name="end_time",
                type="string",
                description="Hora de fin ISO format",
                required=False
            )
        ]

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

        try:
            # Formato: khal new START [END] TITLE
            # Ejemplo: khal new 2024-12-31 14:00 16:00 "Reunión importante"

            start_dt = datetime.fromisoformat(start_time)

            cmd = ["khal", "new"]

            # Fecha y hora de inicio
            cmd.append(start_dt.strftime("%Y-%m-%d"))
```

```
            cmd.append(start_dt.strftime("%H:%M"))

            # Hora de fin (si se especifica)
            if end_time:
                end_dt = datetime.fromisoformat(end_time)
                cmd.append(end_dt.strftime("%H:%M"))

            # Título
            cmd.append(title)

            result = subprocess.run(
                cmd,
                capture_output=True,
                text=True,
                check=True
            )

            return {
                "success": True,
                "message": f"Evento '{title}' creado en Khal",
                "output": result.stdout
            }

        except subprocess.CalledProcessError as e:
            return {
                "success": False,
                "error": f"Error ejecutando khal: {e.stderr}"
            }
        except Exception as e:
            return {
                "success": False,
                "error": f"Error: {str(e)}"
            }
```

Comandos útiles de Khal:

```
# Crear evento
khal new 2024-12-31 14:00 16:00 "Reunión"

# Crear evento de día completo
khal new 2024-12-31 "Cumpleaños de Juan"

# Listar eventos
khal list today
khal list tomorrow
khal list 2024-12-31

# Buscar eventos
khal search "reunión"

# Ver agenda
khal calendar
```

Usando librería Python (icalendar)

Si Khal no está instalado o quieres más control:

```
# Instalar: uv add icalendar

from icalendar import Calendar, Event
from datetime import datetime
import uuid

class KhalICalTool(Tool):
    def __init__(self):
        # Khal guarda calendarios en ~/.local/share/khal/calendars/
        self.calendar_dir = Path.home() / ".local" / "share" / "khal" /
"calendars" / "default"

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

        # Crear evento iCal
        cal = Calendar()
        event = Event()

        event.add('summary', title)
        event.add('dtstart', datetime.fromisoformat(start_time))
        event.add('uid', str(uuid.uuid4()))
        event.add('dtstamp', datetime.now())

        cal.add_component(event)

        # Guardar en archivo .ics
        event_file = self.calendar_dir / f"{uuid.uuid4()}.ics"
        with open(event_file, 'wb') as f:
            f.write(cal.to_ical())

        return {
            "success": True,
            "message": f"Evento '{title}' creado en Khal"
        }
```

Obsidian

Obsidian es un editor de notas en Markdown. La integración es muy simple: escribir archivos `.md`.

Crear notas en Obsidian

```python
# src/tools/obsidian_tool.py
from pathlib import Path
from datetime import datetime
from typing import Dict, Any, List
from .base import Tool, ToolParameter

class ObsidianCreateNoteTool(Tool):
    def __init__(self):
        # Ruta a tu vault de Obsidian (configurable)
        # Puedes ponerla en el .env como OBSIDIAN_VAULT_PATH
        self.vault_path = Path.home() / "Documents" / "ObsidianVault"

        # O desde settings
        from ..utils.config import get_settings
        settings = get_settings()
        # Agrega OBSIDIAN_VAULT_PATH a tu .env
        # self.vault_path = Path(os.getenv("OBSIDIAN_VAULT_PATH",
self.vault_path))

    @property
    def name(self) -> str:
        return "obsidian_create_note"

    @property
    def description(self) -> str:
        return (
            "Crea una nota nueva en Obsidian. "
            "Úsala cuando el usuario quiera tomar una nota, "
            "guardar información, o crear un documento."
        )

    @property
    def parameters(self) -> List[ToolParameter]:
        return [
            ToolParameter(
                name="title",
                type="string",
                description="Título de la nota",
                required=True
            ),
            ToolParameter(
                name="content",
                type="string",
                description="Contenido de la nota en Markdown",
                required=True
            ),
            ToolParameter(
                name="folder",
                type="string",
                description="Carpeta donde crear la nota (opcional)",
                required=False
            ),
            ToolParameter(
```

```
                name="tags",
                type="array",
                description="Lista de tags para la nota",
                required=False
            )
        ]

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

        try:
            # Determinar ruta completa
            if folder:
                note_dir = self.vault_path / folder
                note_dir.mkdir(parents=True, exist_ok=True)
            else:
                note_dir = self.vault_path

            # Nombre del archivo (sanitizar título)
            safe_title = title.replace("/", "-").replace("\\", "-")
            note_file = note_dir / f"{safe_title}.md"

            # Construir contenido con frontmatter (metadata de Obsidian)
            frontmatter = "---\n"
            frontmatter += f"created: {datetime.now().isoformat()}\n"
            if tags:
                frontmatter += f"tags: {', '.join(tags)}\n"
            frontmatter += "---\n\n"

            full_content = f"{frontmatter}# {title}\n\n{content}\n"

            # Escribir archivo
            with open(note_file, "w", encoding="utf-8") as f:
                f.write(full_content)

            return {
                "success": True,
                "message": f"Nota '{title}' creada en Obsidian",
                "path": str(note_file.relative_to(self.vault_path))
            }

        except Exception as e:
            return {
                "success": False,
                "error": f"Error creando nota: {str(e)}"
            }
```

Buscar en notas de Obsidian

```python
class ObsidianSearchTool(Tool):
    @property
    def name(self) -> str:
        return "obsidian_search"

    @property
    def description(self) -> str:
        return "Busca texto en las notas de Obsidian"

    @property
    def parameters(self) -> List[ToolParameter]:
        return [
            ToolParameter(
                name="query",
                type="string",
                description="Texto a buscar",
                required=True
            )
        ]

    async def execute(self, **kwargs) -> Dict[str, Any]:
        query = kwargs.get("query")
        vault_path = Path.home() / "Documents" / "ObsidianVault"

        results = []

        # Buscar en todos los archivos .md
        for note_file in vault_path.rglob("*.md"):
            try:
                with open(note_file, "r", encoding="utf-8") as f:
                    content = f.read()

                    if query.lower() in content.lower():
                        # Encontrado - extraer contexto
                        lines = content.split("\n")
                        matching_lines = [
                            line for line in lines
                            if query.lower() in line.lower()
                        ]

                        results.append({
                            "file": note_file.name,
                            "path": str(note_file.relative_to(vault_path)),
                            "matches": matching_lines[:3]  # Primeras 3
coincidencias
                        })
            except Exception:
                continue

        return {
            "success": True,
            "results": results,
```

```
            "count": len(results)
        }
```

Estructura típica de Obsidian:

```
~/Documents/ObsidianVault/
├── Daily Notes/
│   ├── 2024-12-31.md
│   └── 2025-01-01.md
├── Projects/
│   ├── Proyecto A.md
│   └── Proyecto B.md
├── Areas/
└── Resources/
```

Navegadores (Firefox/Brave)

Puedes controlar navegadores de varias formas:

Opción A: Abrir URLs (Simple)

```python
# src/tools/browser_tool.py
import webbrowser
import subprocess
from typing import Dict, Any, List
from .base import Tool, ToolParameter

class BrowserOpenTool(Tool):
    @property
    def name(self) -> str:
        return "browser_open"

    @property
    def description(self) -> str:
        return (
            "Abre una URL en el navegador web. "
            "Úsala cuando el usuario pida abrir una página web, "
            "buscar algo en internet, o visitar un sitio."
        )

    @property
    def parameters(self) -> List[ToolParameter]:
        return [
            ToolParameter(
                name="url",
                type="string",
                description="URL completa a abrir (ej:
https://google.com)",
```

```
                required=True
            ),
            ToolParameter(
                name="browser",
                type="string",
                description="Navegador a usar",
                required=False,
                enum=["firefox", "brave", "chromium", "default"]
            )
        ]

    async def execute(self, **kwargs) -> Dict[str, Any]:
        url = kwargs.get("url")
        browser = kwargs.get("browser", "default")

        try:
            # Si no es una URL completa, asume búsqueda de Google
            if not url.startswith(("http://", "https://")):
                url = f"https://www.google.com/search?q={url.replace(' ',
'+')}"

            if browser == "default":
                # Usar navegador por defecto del sistema
                webbrowser.open(url)
            elif browser == "firefox":
                subprocess.Popen(["firefox", url])
            elif browser == "brave":
                subprocess.Popen(["brave", url])
            elif browser == "chromium":
                subprocess.Popen(["chromium", url])

            return {
                "success": True,
                "message": f"Abriendo {url} en {browser}",
                "url": url
            }

        except Exception as e:
            return {
                "success": False,
                "error": f"Error abriendo navegador: {str(e)}"
            }
```

Opción B: Control avanzado (Selenium)

Para control completo del navegador (hacer clicks, llenar formularios, etc.):

```
# Instalar: uv add selenium

from selenium import webdriver
from selenium.webdriver.firefox.options import Options as FirefoxOptions
```

```
class BrowserAutomationTool(Tool):
    def __init__(self):
        # Configurar navegador en modo headless (sin ventana)
        self.options = FirefoxOptions()
        self.options.add_argument("--headless")
        self.driver = None

    async def execute(self, **kwargs) -> Dict[str, Any]:
        url = kwargs.get("url")
        action = kwargs.get("action", "open")  # open, screenshot,
extract_text

        try:
            if not self.driver:
                self.driver = webdriver.Firefox(options=self.options)

            self.driver.get(url)

            if action == "screenshot":
                screenshot_path =
f"/tmp/screenshot_{datetime.now().timestamp()}.png"
                self.driver.save_screenshot(screenshot_path)
                return {
                    "success": True,
                    "screenshot": screenshot_path
                }

            elif action == "extract_text":
                text = self.driver.find_element("tag name", "body").text
                return {
                    "success": True,
                    "text": text[:500]  # Primeros 500 caracteres
                }

            return {
                "success": True,
                "message": f"Página {url} cargada"
            }

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

Opción C: Marcadores del navegador

Leer marcadores de Firefox:

```python
import sqlite3
from pathlib import Path

class FirefoxBookmarksTool(Tool):
    def __init__(self):
        # Firefox guarda marcadores en places.sqlite
        firefox_profile = Path.home() / ".mozilla" / "firefox"
        # Encontrar el perfil por defecto
        profiles = list(firefox_profile.glob("*.default-release"))
        if profiles:
            self.bookmarks_db = profiles[0] / "places.sqlite"

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

        try:
            conn = sqlite3.connect(self.bookmarks_db)
            cursor = conn.cursor()

            # Buscar marcadores
            sql = """
                SELECT moz_bookmarks.title, moz_places.url
                FROM moz_bookmarks
                JOIN moz_places ON moz_bookmarks.fk = moz_places.id
                WHERE moz_bookmarks.title LIKE ?
            """

            cursor.execute(sql, (f"%{query}%",))
            results = cursor.fetchall()

            conn.close()

            bookmarks = [
                {"title": title, "url": url}
                for title, url in results
            ]

            return {
                "success": True,
                "bookmarks": bookmarks,
                "count": len(bookmarks)
            }

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

Logseq

Logseq es similar a Obsidian pero usa un formato de bloques.

Crear páginas en Logseq

```python
# src/tools/logseq_tool.py
from pathlib import Path
from datetime import datetime
from typing import Dict, Any, List
from .base import Tool, ToolParameter

class LogseqCreatePageTool(Tool):
    def __init__(self):
        # Ruta a tu grafo de Logseq
        self.graph_path = Path.home() / "Documents" / "LogseqGraph"
        self.pages_dir = self.graph_path / "pages"
        self.journals_dir = self.graph_path / "journals"

    @property
    def name(self) -> str:
        return "logseq_create_page"

    @property
    def description(self) -> str:
        return "Crea una página nueva en Logseq"

    @property
    def parameters(self) -> List[ToolParameter]:
        return [
            ToolParameter(
                name="title",
                type="string",
                description="Título de la página",
                required=True
            ),
            ToolParameter(
                name="content",
                type="string",
                description="Contenido de la página",
                required=True
            ),
            ToolParameter(
                name="tags",
                type="array",
                description="Tags para la página",
                required=False
            )
        ]

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

```
        try:
            # Sanitizar título para nombre de archivo
            safe_title = title.replace("/", "___").replace("\\", "___")
            page_file = self.pages_dir / f"{safe_title}.md"

            # Formato de Logseq: bullet points con - o •
            logseq_content = f"- {content}\n"

            # Agregar tags
            if tags:
                tags_str = " ".join([f"#{tag}" for tag in tags])
                logseq_content += f"  tags:: {tags_str}\n"

            # Escribir archivo
            with open(page_file, "w", encoding="utf-8") as f:
                f.write(logseq_content)

            return {
                "success": True,
                "message": f"Página '{title}' creada en Logseq",
                "path": str(page_file.name)
            }

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

Agregar a journal diario

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

    @property
    def description(self) -> str:
        return "Agrega una entrada al journal diario de Logseq"

    @property
    def parameters(self) -> List[ToolParameter]:
        return [
            ToolParameter(
                name="content",
                type="string",
                description="Contenido a agregar",
                required=True
            )
        ]
```

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

        try:
            # Logseq usa formato YYYY_MM_DD para journals
            today = datetime.now().strftime("%Y_%m_%d")
            journal_file = self.journals_dir / f"{today}.md"

            # Crear archivo si no existe
            if not journal_file.exists():
                with open(journal_file, "w") as f:
                    f.write(f"- {datetime.now().strftime('%A, %B %d,
%Y')}\n")

            # Agregar entrada
            with open(journal_file, "a", encoding="utf-8") as f:
                timestamp = datetime.now().strftime("%H:%M")
                f.write(f"- {timestamp} - {content}\n")

            return {
                "success": True,
                "message": f"Entrada agregada al journal de hoy"
            }

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

Aplicaciones CLI Genéricas

Para cualquier aplicación de línea de comandos:

```
# src/tools/cli_tool.py
import subprocess
from typing import Dict, Any, List
from .base import Tool, ToolParameter

class GenericCLITool(Tool):
    """Herramienta genérica para ejecutar comandos CLI"""

    @property
    def name(self) -> str:
        return "run_cli_command"

    @property
    def description(self) -> str:
        return (
```

```
            "Ejecuta un comando de línea de comandos. "
            "CUIDADO: Solo usar para comandos seguros y de confianza."
        )

    @property
    def parameters(self) -> List[ToolParameter]:
        return [
            ToolParameter(
                name="command",
                type="string",
                description="Comando a ejecutar",
                required=True
            ),
            ToolParameter(
                name="args",
                type="array",
                description="Argumentos del comando",
                required=False
            )
        ]

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

        # IMPORTANTE: Validación de seguridad
        # Whitelist de comandos permitidos
        allowed_commands = [
            "calcurse", "khal", "ls", "cat", "grep",
            "firefox", "brave", "code", "vim"
        ]

        if command not in allowed_commands:
            return {
                "success": False,
                "error": f"Comando '{command}' no permitido por seguridad"
            }

        try:
            result = subprocess.run(
                [command] + args,
                capture_output=True,
                text=True,
                timeout=10,  # Timeout de 10 segundos
                check=True
            )

            return {
                "success": True,
                "output": result.stdout,
                "command": f"{command} {' '.join(args)}"
            }

        except subprocess.TimeoutExpired:
```

```
            return {
                "success": False,
                "error": "Comando excedió el tiempo límite"
            }
        except subprocess.CalledProcessError as e:
            return {
                "success": False,
                "error": f"Error ejecutando comando: {e.stderr}"
            }
        except Exception as e:
            return {
                "success": False,
                "error": f"Error: {str(e)}"
            }
```

Tips Generales de Integración

1. Configuración en .env

Agrega rutas configurables:

```
# .env
OBSIDIAN_VAULT_PATH=/home/kiki/Documents/ObsidianVault
LOGSEQ_GRAPH_PATH=/home/kiki/Documents/LogseqGraph
CALCURSE_DIR=/home/kiki/.local/share/calcurse
BROWSER_DEFAULT=firefox
```

Luego en tu código:

```
import os
from pathlib import Path

vault_path = Path(os.getenv("OBSIDIAN_VAULT_PATH",
"~/Documents/ObsidianVault"))
```

2. Manejo de errores robusto

Siempre valida que las aplicaciones estén instaladas:

```
import shutil

def check_command_exists(command):
    """Verifica si un comando está disponible"""
    return shutil.which(command) is not None

# En tu herramienta
```

```
    if not check_command_exists("calcurse"):
        return {
            "success": False,
            "error": "Calcurse no está instalado. Instala con: sudo pacman -S
    calcurse"
        }
```

3. Logging para debugging

```
import logging
logger = logging.getLogger(__name__)

logger.info(f"Ejecutando comando: {command}")
logger.debug(f"Argumentos: {args}")
logger.error(f"Error: {e}")
```

4. Tests unitarios

```
# tests/test_calcurse_tool.py
import pytest
from src.tools.calcurse_tool import CalcurseCreateEventTool

@pytest.mark.asyncio
async def test_create_event():
    tool = CalcurseCreateEventTool()

    result = await tool.execute(
        title="Test Event",
        start_time="2025-01-01T10:00:00",
        duration_minutes=60
    )

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

Ejemplos de Uso Final

Una vez implementado, el usuario puede hacer cosas como:

```
Usuario: "Abre YouTube en Firefox"
→ browser_open se ejecuta con url="youtube.com" y browser="firefox"

Usuario: "Crea una nota en Obsidian sobre lo que hablamos"
→ obsidian_create_note se ejecuta con el resumen de la conversación

Usuario: "Agenda reunión en mi calendario para mañana a las 3"
```

```
→ calcurse_create_event o khal_create_event (el que tengas configurado)

Usuario: "Busca en mis notas algo sobre Python"
→ obsidian_search con query="Python"
```

Recursos Adicionales

- **Calcurse**: `man calcurse`, `/usr/share/doc/calcurse/`
- **Khal**: Documentación oficial
- **Obsidian**: Forum, Plugins
- **Logseq**: Documentación
- **Selenium**: Python bindings

¿Preguntas? ¿Quieres un ejemplo más específico de alguna integración?