GitHub Actions: Variables de Contexto

Tecnologías de Servicios para Ciencia de Datos

Universidad de Las Palmas

Introducción

GitHub Actions es una plataforma que permite automatizar flujos de trabajo directamente desde un repositorio de GitHub. Durante la ejecución de estos flujos de trabajo, es común necesitar acceso a información dinámica sobre el entorno, los trabajos en curso, las entradas proporcionadas y otros elementos.

Para facilitar esta interacción, GitHub Actions proporciona un conjunto de variables de contexto predefinidas. Estas variables ofrecen datos clave sobre el flujo de trabajo, los pasos, los secretos, la configuración del entorno, entre otros. Al comprender cómo funcionan estas variables y cómo pueden ser utilizadas, los desarrolladores pueden construir flujos de trabajo más flexibles y adaptativos.

En este documento, se describen las principales variables de contexto disponibles en GitHub Actions, junto con sus tipos y funcionalidades. Cada sección está dedicada a una variable específica, con explicaciones detalladas y enlaces a la documentación oficial para mayor referencia.

1. `github`

- **Tipo:** `objeto`
- **Descripción:** La variable `github` contiene información detallada sobre la ejecución del flujo de trabajo actual. Incluye datos sobre el repositorio, el evento que activó la ejecución, los actores involucrados y otros metadatos importantes.
- **Campos principales y ejemplos:**
  - `github.repository`:

- **Descripción:** Proporciona el nombre completo del repositorio en el formato `usuario/repositorio`.
- **Uso:** Se puede utilizar para registrar información sobre el repositorio o para configuraciones específicas en scripts personalizados.
- **Ejemplo:**

```
steps:
  - name: Mostrar el nombre del repositorio
    run: echo "${{ github.repository }}"
```

- `github.event_name`:
  - **Descripción:** Indica el evento que activó el flujo de trabajo (por ejemplo, `push`, `pull_request`).
  - **Uso:** Útil para personalizar las acciones según el evento desencadenante.
  - **Ejemplo:**

```
steps:
  - name: Detectar evento
    run: echo "${{ github.event_name }}"
```

- `github.actor`:
  - **Descripción:** Nombre del usuario que inició el evento.
  - **Uso:** Puede utilizarse para enviar notificaciones personalizadas o registrar información del autor.
  - **Ejemplo:**

```
steps:
  - name: Notificar al autor
    run: echo "Gracias, ${{ github.actor }}"
```

- `github.ref`:
  - **Descripción:** Proporciona la referencia de la rama o etiqueta asociada al evento.
  - **Uso:** Permite realizar acciones específicas dependiendo de la rama o etiqueta.
  - **Ejemplo:**

```
steps:
  - name: Verificar la rama activa
    run: echo "${{ github.ref }}"
```

- `github.sha`:
  - **Descripción:** Hash de confirmación SHA del commit asociado al evento.

- **Uso:** Se utiliza para identificar de manera única el commit en scripts o registros.
- **Ejemplo:**

```
steps:
  - name: Mostrar el SHA del commit
    run: echo "El SHA del commit es ${{ github.sha }}"
```

- `github.workflow`:
  - **Descripción:** Proporciona el nombre del flujo de trabajo en ejecución.
  - **Uso:** Útil para registrar o reportar información del flujo de trabajo actual.
  - **Ejemplo:**

```
steps:
  - name: Reportar el nombre del flujo de trabajo
    run: echo "Flujo ${{ github.workflow }}"
```

- `github.run_id` y `github.run_number`:
  - **Descripción:** Identificadores únicos de la ejecución del flujo de trabajo.
  - **Uso:** Útiles para generar reportes únicos o referenciar ejecuciones específicas.
  - **Ejemplo:**

```
steps:
  - name: Identificar ejecución
    run: echo "Ejecución ID: ${{ github.run_id }},
    Número: ${{ github.run_number }}"
```

2. env

- **Tipo:** `objeto`
- **Descripción:** La variable `env` contiene las variables de entorno configuradas en el flujo de trabajo, el trabajo o el paso actual. Estas variables se pueden definir de forma global en el archivo del flujo de trabajo o a nivel local en un paso específico.
- **Uso común:**
  - Personalización de comandos con variables específicas.
  - Almacenamiento de configuraciones reutilizables, como nombres de archivos, rutas o claves no sensibles.

- Acceso a variables compartidas entre pasos.

- **Cómo definir variables:** Las variables de entorno pueden definirse en la sección `env` a nivel de flujo de trabajo, trabajo o paso. Ejemplo:

```
env:
  MY_ENV_VAR: "Valor global"
```

- **Ejemplos prácticos:**

  - **Definir una variable global y usarla en un paso:**

```
name: Uso de variables de entorno
on: [push]
jobs:
  example:
    runs-on: ubuntu-latest
    env:
      GREETING: "Hola, mundo"
    steps:
      - name: Usar la variable de entorno
        run: echo "${{ env.GREETING }}"
```

  - **Definir una variable en un paso específico:**

```
steps:
  - name: Definir variable de entorno
    env:
      CUSTOM_PATH: "/mi/directorio"
    run: echo "Ruta personalizada: $CUSTOM_PATH"
```

  - **Usar una variable en scripts:**

```
steps:
  - name: Usar una variable en un script
    env:
      FILE_NAME: "resultado.txt"
    run: |
      echo "Creando archivo con nombre $FILE_NAME"
      touch $FILE_NAME
```

3. vars

- **Tipo:** `objeto`

- **Descripción:** La variable `vars` contiene las variables configuradas a nivel de repositorio, organización o entorno. Estas variables son útiles para almacenar valores reutilizables que pueden ser utilizados en múltiples flujos de trabajo dentro del repositorio o en varios repositorios dentro de una organización.
- **Uso común:**
  - Definición de valores globales reutilizables, como URLs de servicios, nombres de equipo o configuraciones predeterminadas.
  - Facilitar la modificación centralizada de valores para múltiples flujos de trabajo.
- **Definir variables a nivel de repositorio:** Para configurar las variables, acceda a la configuración del repositorio en GitHub:
  - Vaya a `Settings >Environments >Add Environment`.
  - Configure variables globales en la sección de `Environment Variables`.
- **Ejemplos prácticos:**
  - **Usar una variable global:**

```
name: Uso de variables globales
on: [push]
jobs:
  example:
    runs-on: ubuntu-latest
    steps:
      - name: Mostrar la variable configurada
        run: echo "Valor de la variable: ${{ vars.MY_GLOBAL_VAR }}"
```

  - **Combinar variables globales con locales:**

```
jobs:
  example:
    runs-on: ubuntu-latest
    steps:
      - name: Combinar variables
        env:
          LOCAL_VAR: "Variable local"
        run: echo "Global: ${{ vars.MY_GLOBAL_VAR }},
          Local: $LOCAL_VAR"
```

- **Utilizar variables en un script:**

```
jobs:
  example:
    runs-on: ubuntu-latest
    steps:
      - name: Usar variable global en un script
        run: |
          echo "Inicio del script"
          echo "El valor configurado es ${{ vars.API_ENDPOINT }}"
```

4. job

- **Tipo:** `objeto`
- **Descripción:** La variable `job` contiene información sobre el trabajo que se está ejecutando actualmente en el flujo de trabajo. Esto incluye el estado, el identificador y otros datos relacionados con la ejecución del trabajo.
- **Campos principales y ejemplos:**
  - `job.status`:
    - **Descripción:** Muestra el estado del trabajo actual (`success`, `failure`, `cancelled`).
    - **Uso:** Útil para realizar acciones condicionales basadas en el resultado del trabajo.
    - **Ejemplo:**

```
steps:
  - name: Verificar el estado del trabajo
    if: ${{ job.status == 'success' }}
    run: echo "El trabajo se ejecutó con éxito."
  - name: Manejar fallos
    if: ${{ job.status == 'failure' }}
    run: echo "El trabajo falló. Revisar los logs."
```

  - `job.name`:
    - **Descripción:** Devuelve el nombre del trabajo actual definido en el flujo de trabajo.
    - **Uso:** Para registrar o mostrar el nombre del trabajo en los logs.
    - **Ejemplo:**

```
steps:
  - name: Mostrar el nombre del trabajo
    run: echo "Nombre del trabajo actual: ${{ job.name }}"
```

- `job.outputs`:
  - **Descripción:** Proporciona acceso a las salidas definidas en el trabajo actual.
  - **Uso:** Permite capturar y usar valores generados en un trabajo para otros trabajos en el flujo de trabajo.
  - **Ejemplo:**

```
jobs:
  generate-output:
    runs-on: ubuntu-latest
    outputs:
      result: ${{ steps.set-output.outputs.result }}
    steps:
      - id: set-output
        run: echo "::set-output name=result::valor_generado"
  use-output:
    runs-on: ubuntu-latest
    needs: generate-output
    steps:
      - name: Usar la salida del trabajo anterior
        run: echo "${{ needs.generate-output.outputs.result }}"
```

5. `jobs`

- **Tipo:** `objeto`
- **Descripción:** La variable `jobs` está disponible únicamente en flujos de trabajo reutilizables. Contiene las salidas (`outputs`) de los trabajos definidos dentro del flujo reutilizable, lo que permite compartir datos entre diferentes trabajos.
- **Uso común:**
  - Acceso a los resultados generados por trabajos previos dentro de un flujo reutilizable.
  - Transmisión de valores entre trabajos relacionados en flujos de trabajo reutilizables.

- **Ejemplo práctico:**
  - **Flujo de trabajo reutilizable con salidas:**

```
# Reusable workflow (reusable.yml)
name: Reusable Workflow
on:
  workflow_call:
    inputs:
      input_value:
        required: true
        type: string
jobs:
  generate-output:
    runs-on: ubuntu-latest
    outputs:
      result: ${{ steps.set-output.outputs.result }}
    steps:
      - id: set-output
        run: echo "::set-output name=result::${{ inputs.input_value }}"
```

  - **Uso del flujo reutilizable:**

```
# Main workflow (main.yml)
name: Main Workflow
on: [push]
jobs:
  use-reusable:
    uses: ./.github/workflows/reusable.yml
    with:
      input_value: "Hola desde el flujo principal"
  show-output:
    runs-on: ubuntu-latest
    needs: use-reusable
    steps:
      - name: Mostrar salida del flujo reutilizable
        run: echo "Resultado: ${{ jobs.use-reusable.outputs.result }}"
```

6. steps

- **Tipo:** `objeto`

- **Descripción:** La variable `steps` contiene información sobre los pasos que ya se han ejecutado dentro del trabajo actual. Esto incluye el estado y las salidas (`outputs`) de cada paso.
- **Uso común:**
  - Capturar los valores generados en pasos anteriores para utilizarlos en pasos posteriores.
  - Verificar el estado de un paso específico antes de continuar con el flujo de trabajo.
- **Campos principales y ejemplos:**
  - `steps.<step-id>.outputs.<output-name>`:
    - **Descripción:** Permite acceder a las salidas de un paso específico, identificándolo por su `id`.
    - **Uso:** Recuperar valores generados en un paso para usarlos en pasos posteriores.
    - **Ejemplo:**

```
steps:
  - id: generate-value
    run: echo "::set-output name=myOutput::Valor generado"
  - name: Usar la salida del paso anterior
    run: echo "${{ steps.generate-value.outputs.myOutput }}"
```

  - `steps.<step-id>.conclusion`:
    - **Descripción:** Devuelve la conclusión de un paso (`success`, `failure`, `cancelled`, etc.).
    - **Uso:** Verificar si un paso anterior se ejecutó correctamente.
    - **Ejemplo:**

```
steps:
  - id: check-status
    run: exit 1
    continue-on-error: true
  - name: Verificar estado
    if: ${{ steps.check-status.conclusion == 'failure' }}
    run: echo "El paso anterior falló."
```

- **Ejemplo completo:**

```
name: Uso de la variable steps
```

```
on: [push]
jobs:
  example:
    runs-on: ubuntu-latest
    steps:
      - id: step-one
        run: echo "::set-output name=data::Hola, mundo"
      - name: Usar salida del primer paso
        run: echo "Salida: ${{ steps.step-one.outputs.data }}"
      - id: check-status
        run: exit 0
      - name: Verificar éxito
        if: ${{ steps.check-status.conclusion == 'success' }}
        run: echo "El paso anterior fue exitoso."
```

7. runner

- **Tipo:** `objeto`
- **Descripción:** La variable `runner` contiene información sobre el entorno del runner donde se está ejecutando el flujo de trabajo. Esto incluye detalles como el nombre del sistema operativo, la arquitectura y el directorio temporal.
- **Campos principales y ejemplos:**
  - `runner.os`:
    - **Descripción:** Especifica el sistema operativo del runner en uso (por ejemplo, `Linux`, `Windows`, `macOS`).
    - **Uso:** Personalizar comandos o scripts según el sistema operativo.
    - **Ejemplo:**

      ```
      steps:
        - name: Mostrar sistema operativo
          run: echo "El sistema operativo del runner es ${{ runner.os }}"
      ```
  - `runner.arch`:
    - **Descripción:** Devuelve la arquitectura de la CPU del runner (por ejemplo, `x64`, `ARM`).
    - **Uso:** Configurar herramientas específicas basadas en la arquitectura.
    - **Ejemplo:**

```
steps:
  - name: Mostrar arquitectura
    run: echo "La arquitectura del runner es ${{ runner.arch }}"
```

- `runner.temp`:
  - **Descripción:** Proporciona la ruta al directorio temporal del runner.
  - **Uso:** Almacenar archivos o datos intermedios de manera temporal durante la ejecución del flujo de trabajo.
  - **Ejemplo:**

```
steps:
  - name: Crear un archivo temporal
    run: echo "Este archivo es temporal" > ${{ runner.temp }}/archivo.txt
  - name: Leer archivo temporal
    run: cat ${{ runner.temp }}/archivo.txt
```

- `runner.tool_cache`:
  - **Descripción:** Devuelve la ruta al directorio donde se almacenan herramientas en caché.
  - **Uso:** Usar herramientas almacenadas en el caché del runner para optimizar la ejecución.
  - **Ejemplo:**

```
steps:
  - name: Ver ruta de herramientas en caché
    run: echo "Directorio de herramientas en caché: ${{ runner.tool_cache }
```

- **Ejemplo completo:**

```
name: Uso de la variable runner
on: [push]
jobs:
  example:
    runs-on: ubuntu-latest
    steps:
      - name: Mostrar información del runner
        run: |
          echo "Sistema operativo: ${{ runner.os }}"
          echo "Arquitectura: ${{ runner.arch }}"
          echo "Directorio temporal: ${{ runner.temp }}"
          echo "Directorio de herramientas en caché: ${{ runner.tool_cache }}"
```

8. secrets

- **Tipo:** `objeto`
- **Descripción:** La variable `secrets` contiene los nombres y valores de secretos configurados en el repositorio o en la organización. Estos secretos se usan para manejar información sensible, como claves API, contraseñas o tokens de acceso, y están encriptados para garantizar su seguridad.
- **Uso común:**
  - Proveer información sensible a los flujos de trabajo sin exponerla en el código fuente.
  - Configurar credenciales para integraciones externas, como despliegues o accesos a bases de datos.
- **Configuración de secretos:**
  - Vaya a la configuración del repositorio en GitHub.
  - Seleccione `Settings >Secrets and variables >Actions`.
  - Agregue un nuevo secreto con un nombre único.
- **Ejemplos prácticos:**
  - **Usar un secreto para autenticación:**

```
steps:
  - name: Conectar con una API
    env:
      API_KEY: ${{ secrets.MY_API_KEY }}
    run: |
      echo "Conectando con la API usando la clave $API_KEY"
```

  - **Despliegue seguro con claves SSH:**

```
steps:
  - name: Configurar clave SSH
    run: |
      mkdir -p ~/.ssh
      echo "${{ secrets.SSH_PRIVATE_KEY }}" > ~/.ssh/id_rsa
      chmod 600 ~/.ssh/id_rsa
  - name: Realizar despliegue
    run: ssh user@server "echo 'Despliegue exitoso'"
```

- **Conexión a una base de datos:**

```
steps:
  - name: Conectar a la base de datos
    env:
      DB_PASSWORD: ${{ secrets.DB_PASSWORD }}
    run: |
      echo "Conectando a la base de datos con contraseña segura"
```

- **Buenas prácticas:**
  - Nunca imprima valores de secretos en los logs del flujo de trabajo.
  - Use secretos exclusivamente para información sensible y no los almacene en variables normales o archivos sin cifrar.
- **Ejemplo completo:**

```
name: Uso de secretos en GitHub Actions
on: [push]
jobs:
  example:
    runs-on: ubuntu-latest
    steps:
      - name: Autenticarse en un servicio
        env:
          SERVICE_TOKEN: ${{ secrets.SERVICE_TOKEN }}
        run: |
          echo "Usando token para autenticación segura"
      - name: Conexión segura a un servidor
        run: |
          ssh -i ${{ secrets.SSH_KEY }} user@server "echo 'Conexión establecid
```

9. strategy

- **Tipo:** `objeto`
- **Descripción:** La variable `strategy` proporciona información sobre la estrategia de ejecución basada en matrices definida en el trabajo actual. Permite ejecutar múltiples configuraciones de manera paralela, basándose en valores predefinidos.
- **Uso común:**

- Ejecutar un trabajo con diferentes configuraciones, como versiones de lenguajes de programación o sistemas operativos.
- Realizar pruebas en múltiples entornos para garantizar la compatibilidad.

- **Definir una estrategia en matrices:**

```
jobs:
  test:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        node-version: [12, 14, 16]
        os: [ubuntu-latest, windows-latest]
```

- **Ejemplos prácticos:**
  - **Usar valores de la matriz en un paso:**

```
jobs:
  test:
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        node-version: [12, 14, 16]
        os: [ubuntu-latest, windows-latest]
    steps:
      - name: Configurar Node.js
        uses: actions/setup-node@v2
        with:
          node-version: ${{ matrix.node-version }}
      - name: Ejecutar pruebas
        run: npm test
```

  - **Acceder a la información de la estrategia:**

```
steps:
  - name: Mostrar información de la estrategia
    run: |
      echo "Sistema operativo: ${{ strategy.os }}"
      echo "Versión de Node.js: ${{ strategy.node-version }}"
```

- **Ejecuciones condicionales:**

- Puede usarse para ejecutar pasos o trabajos específicos dependiendo de los valores de la estrategia.

```
steps:
  - name: Ejecutar en sistemas Linux
    if: ${{ matrix.os == 'ubuntu-latest' }}
    run: echo "Este paso solo se ejecuta en Linux"
```

- **Ejemplo completo:**

```
name: Pruebas con diferentes configuraciones
on: [push]
jobs:
  test:
    strategy:
      matrix:
        node-version: [12, 14, 16]
        os: [ubuntu-latest, macos-latest]
    runs-on: ${{ matrix.os }}
    steps:
      - name: Configurar Node.js
        uses: actions/setup-node@v2
        with:
          node-version: ${{ matrix.node-version }}
      - name: Ejecutar pruebas
        run: npm test
      - name: Mostrar configuración
        run: |
          echo "Sistema operativo: ${{ matrix.os }}"
          echo "Versión de Node.js: ${{ matrix.node-version }}"
```

10. `matrix`

- **Tipo:** `objeto`
- **Descripción:** La variable `matrix` contiene las propiedades de la matriz definidas en el flujo de trabajo que se aplican al trabajo actual. Esto es útil para ejecutar trabajos con múltiples configuraciones paralelas, como diferentes sistemas operativos, versiones de lenguajes o configuraciones personalizadas.
- **Uso común:**

- Configuración y prueba de software en diferentes entornos.
- Definición de matrices con múltiples combinaciones de valores.
- Uso de valores dinámicos para personalizar la ejecución de trabajos.

- **Definir una matriz:**

```
jobs:
  build:
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest, macos-latest]
        node-version: [12, 14, 16]
```

- **Ejemplos prácticos:**

  - **Usar valores de la matriz en pasos:**

```
jobs:
  build:
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest]
        node-version: [12, 16]
    steps:
      - name: Configurar Node.js
        uses: actions/setup-node@v2
        with:
          node-version: ${{ matrix.node-version }}
      - name: Ejecutar pruebas
        run: npm test
      - name: Mostrar configuración actual
        run: echo "OS: ${{ matrix.os }}, Node.js: ${{ matrix.node-version }}"
```

  - **Ejecutar acciones condicionales:**

```
steps:
  - name: Ejecutar paso solo en Ubuntu
    if: ${{ matrix.os == 'ubuntu-latest' }}
    run: echo "Esto se ejecuta solo en Ubuntu"
```

- **Ejecuciones avanzadas con exclusiones:** Si ciertas combinaciones de valores en la matriz no deben ejecutarse, se pueden excluir explícitamente:

```
jobs:
  build:
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest]
        node-version: [12, 14]
      exclude:
        - os: windows-latest
          node-version: 12
    runs-on: ${{ matrix.os }}
    steps:
      - name: Mostrar configuración
        run: echo "OS: ${{ matrix.os }}, Node.js: ${{ matrix.node-version }}"
```

- **Ejemplo completo:**

```
name: Pruebas con matriz
on: [push]
jobs:
  test:
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest]
        python-version: [3.8, 3.9]
    runs-on: ${{ matrix.os }}
    steps:
      - name: Configurar Python
        uses: actions/setup-python@v2
        with:
          python-version: ${{ matrix.python-version }}
      - name: Ejecutar pruebas
        run: python -m unittest
      - name: Mostrar configuración
        run: echo "OS: ${{ matrix.os }}, Python: ${{ matrix.python-version }}"
```

11. needs

- **Tipo:** `objeto`

- **Descripción:** La variable `needs` contiene las salidas (`outputs`) de todos los trabajos que están definidos como dependencias (`needs`) del trabajo actual. Esto permite que un trabajo acceda a los resultados generados por trabajos previos.
- **Uso común:**
  - Pasar datos entre trabajos dependientes.
  - Ejecutar acciones basadas en los resultados de trabajos previos.
  - Crear flujos de trabajo dinámicos que dependan de cálculos o salidas previas.
- **Ejemplo de configuración de dependencias:**

```
jobs:
  job1:
    runs-on: ubuntu-latest
    steps:
      - id: generate-output
        run: echo "::set-output name=result::Resultado de Job1"

  job2:
    runs-on: ubuntu-latest
    needs: job1
    steps:
      - name: Usar salida de job1
        run: echo "Salida de Job1: ${{ needs.job1.outputs.result }}"
```

- **Ejemplos prácticos:**
  - **Acceso a salidas de múltiples trabajos:**

```
jobs:
  job1:
    runs-on: ubuntu-latest
    steps:
      - id: output1
        run: echo "::set-output name=value::Valor desde Job1"

  job2:
    runs-on: ubuntu-latest
    steps:
```

```
      - id: output2
        run: echo "::set-output name=value::Valor desde Job2"

  aggregate:
    runs-on: ubuntu-latest
    needs: [job1, job2]
    steps:
      - name: Agregar salidas
        run: |
          echo "Resultado de Job1: ${{ needs.job1.outputs.value }}"
          echo "Resultado de Job2: ${{ needs.job2.outputs.value }}"
```

- **Condiciones basadas en resultados previos:**

```
jobs:
  job1:
    runs-on: ubuntu-latest
    steps:
      - id: generate-output
        run: echo "::set-output name=result::success"

  job2:
    runs-on: ubuntu-latest
    needs: job1
    steps:
      - name: Ejecutar si Job1 fue exitoso
        if: ${{ needs.job1.outputs.result == 'success' }}
        run: echo "Job1 fue exitoso. Continuando..."
```

- **Ejemplo completo:**

```
name: Uso de la variable needs
on: [push]
jobs:
  setup:
    runs-on: ubuntu-latest
    steps:
      - id: prepare
        run: echo "::set-output name=config::Configuración completada"

  process:
    runs-on: ubuntu-latest
```

```
    needs: setup
    steps:
      - name: Usar configuración de setup
        run: echo "Configuración: ${{ needs.setup.outputs.config }}"
  finalize:
    runs-on: ubuntu-latest
    needs: process
    steps:
      - name: Finalizar flujo de trabajo
        run: echo "Flujo de trabajo completo usando salida de todos los trabaj
```

12. inputs

- **Tipo:** `objeto`
- **Descripción:** La variable `inputs` contiene los valores de las entradas (`inputs`) proporcionadas a un flujo de trabajo reutilizable o a un flujo de trabajo manual desencadenado por el usuario.
- **Uso común:**
  - Personalizar la ejecución de flujos de trabajo reutilizables.
  - Proporcionar datos específicos al iniciar un flujo de trabajo manualmente.
- **Definir entradas en un flujo de trabajo reutilizable:**

```
name: Reusable Workflow
on:
  workflow_call:
    inputs:
      input_value:
        required: true
        type: string
jobs:
  example:
    runs-on: ubuntu-latest
    steps:
      - name: Mostrar el valor de la entrada
        run: echo "El valor de la entrada es ${{ inputs.input_value }}"
```

- **Ejemplo de uso de entradas:**

- **Proporcionar entradas al usar un flujo reutilizable:**

```
name: Main Workflow
on: [push]
jobs:
  use-reusable:
    uses: ./.github/workflows/reusable.yml
    with:
      input_value: "Hola, mundo"
```

- **Usar entradas en flujos manuales:**

```
name: Manual Workflow
on:
  workflow_dispatch:
    inputs:
      username:
        description: "Nombre de usuario"
        required: true
        default: "Invitado"
jobs:
  example:
    runs-on: ubuntu-latest
    steps:
      - name: Mostrar nombre de usuario
        run: echo "Hola, ${{ inputs.username }}"
```

- **Ejemplo completo de entradas manuales:**

```
name: Workflow Dispatch
on:
  workflow_dispatch:
    inputs:
      environment:
        description: "Entorno de despliegue"
        required: true
        default: "producción"
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Desplegar en el entorno especificado
        run: echo "Desplegando en ${{ inputs.environment }}"
```

- **Validaciones comunes:**
  - Puede usarse para verificar o condicionar los valores de entrada:

```
steps:
  - name: Verificar el entorno
    if: ${{ inputs.environment == 'producción' }}
    run: echo "Desplegando en producción"
```