

GitIssue-Manager — Guia (PT-BR)

Contents

GitIssue-Manager – Guia Didático para Leigos (Português)

****Bem-vindo!**** Esse guia explica de forma simples e visual como funciona a ferramenta, mes

0 Problema: Por que essa ferramenta existe?

Imagine que você está planejando um novo sprint com 20 tarefas:

[X] Forma antiga (manual)

\\\

1. *Você escreve em um documento Markdown: "FEATURE-001: Login do usuário"*
 2. *Abre o GitHub*
 3. *Vai para o tab Issues*
 4. *Clica em "New Issue"*
 5. *Copia o título do documento para a issue*
 6. *Copia a descrição manualmente*
 7. *Adiciona labels manualmente*
 8. *Anexa à board ProjectV2*
 9. *Repete isso 20 vezes...*
- [TIME] Resultado: 30–60 minutos gastos apenas criando issues*

\\\

[OK] Forma inteligente (GitIssue-Manager)

\\\

1. *Você escreve em um documento Markdown: "FEATURE-001: Login do usuário"*
 2. *Executa: node client/prepare.js*
 3. *Executa: node server/executor.js --execute*
 4. *PRONTO! Todas as 20 issues foram criadas automaticamente no GitHub*
- [TIME] Resultado: 2 minutos (ou menos!)*

\\\

0 que a ferramenta faz (em termos simples)

GitIssue-Manager é um ****assistente automático**** que:

1. ****Lê**** seu arquivo de planejamento (SPRINT.md)
2. ****Entende**** a estrutura (tarefas, subtarefas, prioridades)

3. **Cria** issues no GitHub automaticamente
4. **Registra** tudo o que faz (para auditoria)

É como se você tivesse um estagiário que:

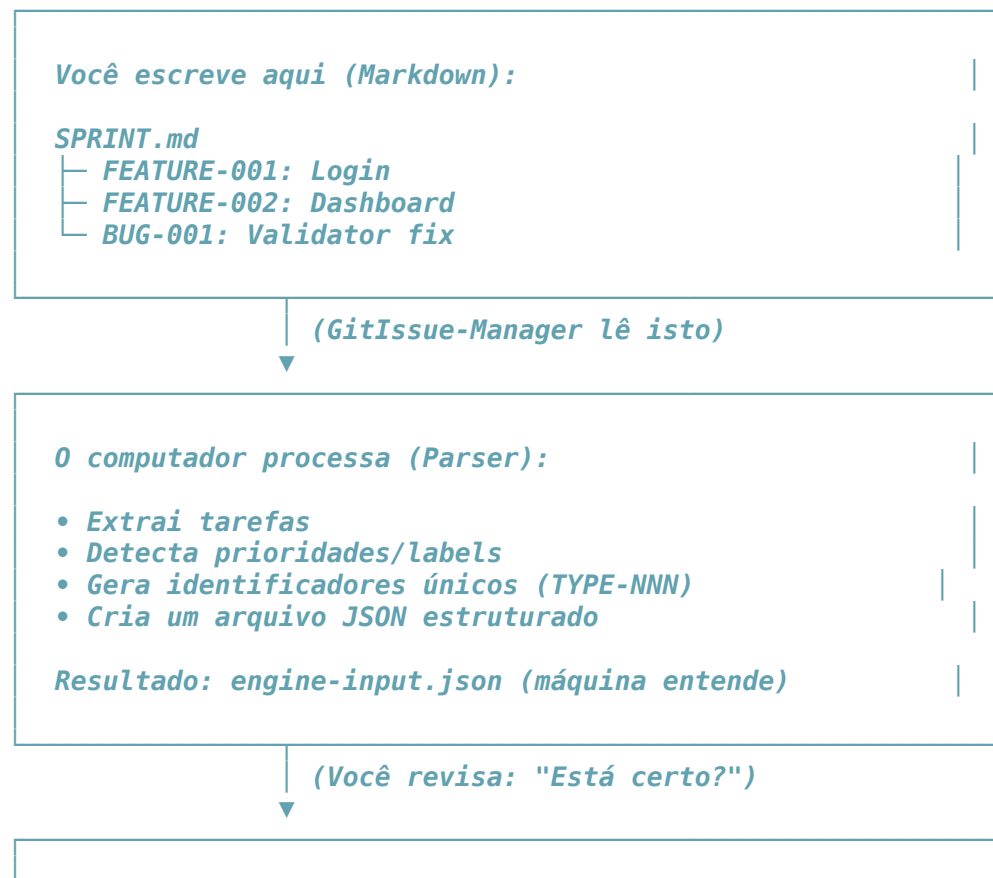
- Copia textos com precisão (sem erros de digitação)
- Nunca esquece de adicionar labels
- Registra exatamente o que fez e quando
- Pode desfazer tudo se errado

Por que é útil?

Cenário	Antes	Depois
-----	-----	-----
Sprint com 20 tarefas	1 hora (manual)	5 minutos (automático)
Atualizar prioridades	Editar cada issue no GitHub	Editar SPRINT.md, sincronizar
Auditoria	"Quem criou essa issue?" → sem registro	<code>logs/audit.jsonl</code> mostra tudo
Erros	"Esqueci de adicionar o label"	Automático, sem erros
Múltiplos repos	Copiar config de um repo para outro	Reutilizar arquivos SPRINT.m

Como funciona (explicado visualmente)

...



PRÉVIA (Dry-Run):
O computador simula criar as issues, mas não as cria no GitHub ainda

Resultado: *logs/dryrun_summary_*.json*
(Você lê: "Sim, está perfeito!")

↓ (Você aprova)
▼

EXECUÇÃO (Real Write):
Agora sim, o computador cria as issues no GitHub

- *Cria issues*
- *Anexa à board ProjectV2*
- *Adiciona labels*
- *Registra tudo em logs/audit.jsonl*



GitHub Issues + ProjectV2 + Audit Log

Tudo criado automaticamente! [OK]

```

---

## ## A Segurança de "Prévia Primeiro"

Um dos poderes dessa ferramenta é a **\*\*prévia (dry-run)\*\***:

1. **\*\*Primeira execução:\*\*** Calcula tudo, mas **NÃO** toca no GitHub
2. **\*\*Você revisa:\*\*** "As tarefas estão certas?"
3. **\*\*Segunda execução:\*\*** Agora sim, cria no GitHub

É como:

- [OK] Draftar um email, reler, depois enviar
- [X] Enviar um email sem ler (erro não pode ser desfeito)

---

## ## Estrutura do documento (SPRINT.md)

Para a ferramenta funcionar bem, seu arquivo precisa ter esta estrutura:

```markdown

Sprint 1: Validação de Endereços

FEATURE-001: Normalizar endereços de validadores

- [] Criar função de normalização
- [] Converter para checksummed format
- [] Testar com dados reais
- [] Integrar nos endpoints da API
- [] Adicionar testes de integração

BUG-001: Bug no validador [priority:high]

- [] Reproduzir o erro
- [] Encontrar a causa
- [] Corrigir
- [] Testar

TASK-001: Documentação

- [] Atualizar README
 - [] Escrever guia de uso
- ``

*****Regras simples:*****

- *`## FEATURE-001:` – início de uma tarefa principal*
- *`- []` – subtarefas dentro da tarefa*
- *`[priority:high]` – opcional, adiciona prioridade*
- *`[labels:backend]` – opcional, adiciona labels*

O que significa "TYPE-NNN"?

*O `TYPE-NNN` é um **identificador único e estável** para cada tarefa.*

- *`FEATURE-001` = tipo é "FEATURE", número é "001"*
- *`BUG-042` = tipo é "BUG", número é "042"*
- *`TASK-005` = tipo é "TASK", número é "005"*

*****Por que é importante?*****

- *Se você mover a tarefa de `SPRINT.md` para `PLAN.md`, a ferramenta ainda reconhece como*
- *Sem ID explícito, a ferramenta cria issues duplicadas (ruim!)*

*****Analogia:*****

- *[OK] Seu CPF (ID) permanece o mesmo mesmo se você mudar de endereço*
- *[X] Sem ID, você seria uma "nova pessoa" a cada vez que se muda*

Passo-a-Passo para o Usuário Final

Pré-requisito

- *Arquivo `SPRINT.md` com suas tarefas (já pronto)*
- *Conversa com alguém técnico para: "rodar a ferramenta"*

O que você faz

- 1. Edita `SPRINT.md` e marca tarefas como `[x]` (feito) ou `[]` (a fazer)*
- 2. Avisa ao time: "Atualizei SPRINT.md"*

0 que o time técnico faz

```bash

# 1. Lê o SPRINT.md

node client/prepare.js ...

# 2. Mostra a prévia para você revisar

node server/executor.js --dry-run

# → Você vê: "Serão criadas 20 issues"

# → Você aprova: "Está certo!"

# 3. Cria no GitHub

node server/executor.js --execute

# → Pronto! Issues no GitHub

```

Você verifica

- Vai para GitHub → Issues tab

- Vê as novas issues com título `[repo-name | #FEATURE-001]`

- Vai para ProjectV2 → vê as issues na board

0 Arquivo de Auditoria (`logs/audit.jsonl`)

Toda vez que a ferramenta cria/atualiza uma issue, ela registra:

```json

{

"timestamp": "2026-01-21T12:34:56Z",

"action": "create\_issue",

"repo": "mzfshark/my-repo",

"issueId": "FEATURE-001",

"title": "[my-repo | #FEATURE-001] Normalize validator addresses",

"status": "success"

}

```

**Por quê?*

- **Compliance:** "Quem fez o quê e quando?"

- **Debugging:** Se algo deu errado, você vê o registro

- **Auditoria:** Registro imutável de todas as ações

Comparação: Ferramenta vs. Manual

Aspecto Manual GitIssue-Manager
----- ----- -----
Tempo (20 issues) 1 hora 5 minutos
Risco de erro Alto (digitação, labels) Baixo (automático)
Atualizar tudo Muito tempo Rápido (refaz tudo)
Auditoria Nenhuma ("quem fez?") Completa (logs)
Aprender curva 0 (UI do GitHub) Baixa (Markdown = fácil)

| Custo | Trabalho manual | 0 (ferramenta reutilizável) |

Próximos Passos (Recomendados)

1. **Fase 1 - Demo:** Time técnico roda a ferramenta em um pequeno exemplo (~5 issues) para
2. **Fase 2 - Piloto:** Use para um sprint real (20–30 issues); validar fluxo
3. **Fase 3 - Produção:** Integre ao processo; documente convenções de nomes (``TYPE-NNN``)
4. **Fase 4 - Automação:** GitHub Actions roda automaticamente quando você faz commit em S

Perguntas Frequentes

P: Posso editar um SPRINT.md existente e sincronizar novamente?

R: Sim! A ferramenta detecta que ``FEATURE-001`` já existe e atualiza a issue em vez de

P: E se eu estragar algo?

R: Use dry-run primeiro (sem risco). Se executar algo errado, é fácil desfazer (deleta

P: Funciona com múltiplos repositórios?

R: Sim! Você pode ter diferentes ``SPRINT.md`` em diferentes repos. A ferramenta process

P: Preciso aprender programação?

R: Não! Você só escreve Markdown (como uma lista com checkboxes). O resto é automático

P: Quem pode usar?

R: Qualquer um que trabalhe com planejamento (product managers, scrum masters, team le

Conclusão

GitIssue-Manager é uma ferramenta que:

- **Economiza tempo:** 1 hora → 5 minutos
- **Reduz erros:** Automático, consistente
- **Melhora rastreabilidade:** Cada mudança registrada
- **É fácil:** Apenas Markdown + dois comandos

Pense nela como um **assistente pessoal automático** que cuida de tarefas chatas para que

Dúvidas? Chama no zap! :)