Guia Prático: Proteção contra Vazamento de Segredos no Git

Visão geral

Este guia configura **quatro camadas de proteção** na máquina de cada desenvolvedor, de forma que qualquer repositório — existente ou futuro — esteja protegido automaticamente. Nenhuma configuração por repositório é necessária.

Camada	Onde atua	O que faz
`.gitignore` global	Antes do `git add`	Impede que arquivos `.env` entrem no staging
Hook pre-commit global (gitleaks)	Antes do `git commit`	Escaneia o conteúdo staged e bloqueia se encontrar segredos
git-cola (interface gráfica)	Durante o staging e commit	Permite revisar visualmente cada diff antes de commitar
Push Protection (GitHub org)	Antes do `git push` chegar ao servidor	Última barreira — GitHub rejeita o push se detectar segredos conhecidos

> **Importante:** A configuração é feita **uma vez por máquina**, não por repositório.

Parte 1: `.gitignore` global

O que é

Um arquivo de ignore que se aplica a **todos os repositórios** da máquina, sem precisar editar o `.gitignore` de cada repo.

Como fazer

```
# Criar o diretório se não existir
mkdir -p ~/.config/git

# Criar/editar o arquivo de ignore global
cat >> ~/.config/git/ignore << 'EOF'

# === Proteção contra vazamento de segredos ===
# Arquivos .env (padrão SciELO e variações comuns)
.envs/
.env
.env.*
.env.local
.env.production
.env.development
.env.staging
*.env

# Chaves e certificados
*.pem
*.key
*.p12
*.pfx
*.jks

# Arquivos de credenciais específicos
.netrc
.pgpass
.my.cnf
credentials.json
service-account*.json
EOF

# Confirmar que o Git sabe onde está o arquivo
git config --global core.excludesFile ~/.config/git/ignore
```

Como testar

```
# 1. Crie um repositório temporário de teste
mkdir /tmp/teste-ignore && cd /tmp/teste-ignore && git init
```

```
# 2. Crie um arquivo .env
echo "DATABASE_PASSWORD=super_secreta_123" > .env

# 3. Tente adicionar ao staging
git add .env

# 4. Verifique — o arquivo NÃO deve aparecer no staging
git status
# Esperado: nada para commitar, .env não aparece

# 5. Teste com subdiretório .envs/
mkdir -p .envs/.local
echo "SECRET_KEY=abc123xyz" > .envs/.local/.django
git add .envs/
git status
# Esperado: nada para commitar

# 6. Limpe
cd ~ && rm -rf /tmp/teste-ignore
```

Resultado esperado: O Git ignora completamente esses arquivos. Eles não aparecem nem em `git status`.

Limitação

Se alguém fizer `git add --force .env`, o `.gitignore` é ignorado. Por isso precisamos da segunda camada.

Parte 2: Hook pre-commit global com gitleaks

O que é

O gitleaks é uma ferramenta que escaneia conteúdo de commits em busca de segredos (senhas, tokens, chaves de API). Configurado como **hook global**, ele roda automaticamente antes de cada `git commit` em qualquer repositório da máquina.

Passo 1: Instalar o gitleaks

```
# === Ubuntu/Debian ===
# Baixar a versão mais recente (verificar em https://github.com/gitleaks/gitleaks/releases)
wget
https://github.com/gitleaks/gitleaks/releases/download/v8.21.2/gitleaks_8.21.2_linux_amd64.deb
sudo dpkg -i gitleaks_8.21.2_linux_amd64.deb

# === macOS ===
brew install gitleaks

# === Verificar instalação ===
gitleaks version
```

Passo 2: Criar o hook pre-commit global

```
# Criar diretório para hooks globais
mkdir -p ~/.git-hooks

# Criar o hook
cat > ~/.git-hooks/pre-commit << 'HOOK'
#!/bin/bash
#
# Hook pre-commit global — gitleaks
# Escaneia o conteúdo staged antes de permitir o commit.
#

# Verificar se gitleaks está instalado
if ! command -v gitleaks &> /dev/null; then
    echo "⚠  gitleaks não encontrado. Instale: https://github.com/gitleaks/gitleaks/releases"
    echo "   Commit permitido, mas SEM proteção contra segredos."
    exit 0
fi

# Definir arquivo de configuração customizado, se existir
CONFIG_FLAG=""
if [ -f "$HOME/.gitleaks.toml" ]; then
    CONFIG_FLAG="--config $HOME/.gitleaks.toml"
```

```
fi

# Escanear apenas o conteúdo staged
gitleaks git --pre-commit --staged --verbose $CONFIG_FLAG

EXIT_CODE=$?

if [ $EXIT_CODE -ne 0 ]; then
    echo ""
    echo "╔══════════════════════════════════════════════════════════════╗"
    echo "║  □ COMMIT BLOQUEADO: segredos detectados!                    ║"
    echo "║                                                              ║"
    echo "║  O que fazer:                                                ║"
    echo "║  1. Remova as senhas/tokens dos arquivos staged              ║"
    echo "║  2. Use variáveis de ambiente ou arquivo .env (não versionado) ║"
    echo "║  3. Rode: gitleaks git --pre-commit --staged --verbose       ║"
    echo "║     para ver exatamente onde estão os segredos               ║"
    echo "╚══════════════════════════════════════════════════════════════╝"
    echo ""
    exit 1
fi
HOOK

# Tornar executável
chmod +x ~/.git-hooks/pre-commit
```

Passo 3: Ativar hooks globais no Git

```
# Esta linha faz o Git usar ~/.git-hooks/ para TODOS os repositórios
git config --global core.hooksPath ~/.git-hooks
```

> **O que acontece com hooks locais?** Quando `core.hooksPath` é definido globalmente, o Git **ignora** hooks locais (`.git/hooks/`). Se algum projeto depende de hooks locais (ex: via husky), veja a seção "Convivência com hooks locais" mais abaixo.

Passo 4 (opcional): Configuração customizada para .env

O gitleaks já detecta padrões conhecidos (tokens AWS, GitHub, Slack, etc.), mas para reforçar a detecção de senhas genéricas em arquivos `.env`:

```
cat > ~/.gitleaks.toml << 'EOF'
[extend]
# Herda todas as ~150 regras padrão do gitleaks
useDefault = true

# Regra adicional: senhas genéricas em arquivos .env
[[rules]]
id = "env-generic-secret"
description = "Valor suspeito atribuído a variável sensível em .env"
regex =
'''(?i)(password|passwd|pwd|secret|token|api_key|apikey|private_key|auth|credential)\s*=\s*['"
]?(\S{8,})['"]?'''
path = '''(\.env|\.env\..+)$'''

# Regra adicional: URLs de banco com senha embutida
[[rules]]
id = "env-database-url"
description = "URL de conexão com senha embutida"
regex = '''(postgres|postgresql|mysql|mongodb|redis|amqp)://[^:]+:[^@]{3,}@'''

[allowlist]
# Ignorar arquivos de exemplo/template
paths = [
    '''\.env\.example$''',
    '''\.env\.sample$''',
    '''\.env\.template$''',
    '''\.env\.dist$''',
]
EOF
```

Como testar

```
# 1. Crie um repositório temporário
mkdir /tmp/teste-gitleaks && cd /tmp/teste-gitleaks && git init

# 2. Crie um arquivo com segredo
cat > config.py << 'EOF'
AWS_SECRET_ACCESS_KEY = "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
DATABASE_URL = "postgres://admin:senha_super_secreta@db.example.com:5432/scielo"
EOF

# 3. Adicione ao staging
git add config.py

# 4. Tente commitar
git commit -m "test: adiciona configurações"

# Esperado: commit BLOQUEADO com mensagem de erro detalhada
# O gitleaks mostra exatamente qual linha e qual padrão foi detectado

# 5. Teste com .env (se criou a config customizada)
cat > .env << 'EOF'
DATABASE_PASSWORD=minha_senha_123
SECRET_KEY=django-insecure-abc123def456
EOF

# Force o add (pula o .gitignore global para testar esta camada)
git add --force .env
git commit -m "test: adiciona .env"
# Esperado: BLOQUEADO

# 6. Teste que commits limpos passam normalmente
cat > readme.md << 'EOF'
# Meu Projeto
Este é um projeto de teste.
EOF

git add readme.md
git commit -m "docs: adiciona readme"
# Esperado: commit realizado com sucesso □
```

```
# 7. Limpe
cd ~ && rm -rf /tmp/teste-gitleaks
```

Convivência com hooks locais (husky, pre-commit framework, etc.)

Se algum projeto usa hooks locais, o `core.hooksPath` global os sobrescreve. Para manter ambos funcionando, modifique o hook global para chamar o hook local se existir:

```
cat > ~/.git-hooks/pre-commit << 'HOOK'
#!/bin/bash

# --- Gitleaks (proteção global) ---
if command -v gitleaks &> /dev/null; then
    CONFIG_FLAG=""
    [ -f "$HOME/.gitleaks.toml" ] && CONFIG_FLAG="--config $HOME/.gitleaks.toml"
    gitleaks git --pre-commit --staged --verbose $CONFIG_FLAG
    if [ $? -ne 0 ]; then
        echo ""
        echo "□ COMMIT BLOQUEADO: segredos detectados!"
        exit 1
    fi
fi

# --- Hook local do projeto (husky, etc.) ---
LOCAL_HOOK=".git/hooks/pre-commit"
if [ -f "$LOCAL_HOOK" ] && [ -x "$LOCAL_HOOK" ]; then
    "$LOCAL_HOOK"
    exit $?
fi

# Se o projeto usa husky (node_modules)
HUSKY_HOOK="node_modules/.husky/pre-commit"
if [ -f "$HUSKY_HOOK" ] && [ -x "$HUSKY_HOOK" ]; then
    "$HUSKY_HOOK"
    exit $?
fi
HOOK
```

```
chmod +x ~/.git-hooks/pre-commit
```

Parte 3: git-cola — revisão visual antes do commit

O que é

O git-cola é uma interface gráfica para Git que facilita a revisão visual de diffs antes de commitar. Ele permite ver exatamente o que está sendo adicionado ao staging, linha por linha, o que ajuda a identificar senhas ou tokens que passaram despercebidos no terminal.

Diferente das camadas anteriores (que são automáticas), o git-cola é uma **ferramenta de apoio visual** — ele não bloqueia nada automaticamente, mas torna muito mais fácil para o desenvolvedor perceber que está prestes a commitar um segredo.

Por que usar

Quando o desenvolvedor trabalha apenas no terminal com `git add .` ou `git add -A`, é fácil adicionar arquivos sem revisar o conteúdo. O git-cola resolve isso porque:

- Mostra o diff colorido de cada arquivo antes de adicionar ao staging
- Permite selecionar quais arquivos (ou até quais linhas) vão para o staging
- Torna visualmente óbvio quando um arquivo `.env` ou uma senha aparece no diff
- Funciona como um "segundo par de olhos" antes das proteções automáticas agirem

Como instalar

```
# === Ubuntu/Debian ===
sudo apt install git-cola

# === macOS ===
brew install git-cola

# === pip (qualquer sistema) ===
pip install git-cola

# === Verificar instalação ===
```

```
git-cola --version
```

Como usar no dia a dia

```
# Abrir git-cola no repositório atual
cd /caminho/do/seu/repo
git-cola
```

O git-cola abre uma janela com quatro painéis principais:

1. **Status** (esquerda superior): lista de arquivos modificados, não rastreados e staged
2. **Diff** (direita): mostra o diff do arquivo selecionado com destaque de cores
3. **Commit** (esquerda inferior): área para escrever a mensagem de commit
4. **Actions**: botões para Stage, Unstage, Commit

Fluxo recomendado:

1. Abra o git-cola no repositório
2. Revise cada arquivo na lista de "Modified" — clique para ver o diff
3. Verifique visualmente se há senhas, tokens ou chaves no diff
4. Selecione apenas os arquivos limpos e clique em "Stage"
5. Escreva a mensagem de commit e clique em "Commit"

Como testar

```
# 1. Crie um repositório temporário
mkdir /tmp/teste-cola && cd /tmp/teste-cola && git init

# 2. Crie um commit inicial
echo "# Projeto teste" > readme.md
git add readme.md && git commit -m "init: readme"

# 3. Crie arquivos com e sem segredo
cat > config.py << 'EOF'
DEBUG = True
ALLOWED_HOSTS = ["*"]
EOF

cat > secrets.py << 'EOF'
DATABASE_PASSWORD = "senha_super_secreta_123"
AWS_KEY = "AKIAIOSFODNN7EXAMPLE"
```

```
EOF

# 4. Abra o git-cola
git-cola

# No git-cola:
# - Clique em "secrets.py" na lista → veja a senha no diff (lado direito)
# - Clique em "config.py" → veja que está limpo
# - Selecione APENAS "config.py" e clique em "Stage"
# - NÃO faça stage do "secrets.py"
# - Escreva a mensagem de commit e commite

# 5. Limpe
cd ~ && rm -rf /tmp/teste-cola
```

Configurar como editor de diff padrão (opcional)

Para abrir o git-cola sempre que quiser revisar antes de commitar:

```
# Criar um alias global para facilitar o uso
git config --global alias.review '!git-cola'
```

Agora, em qualquer repositório:

```
git review   # abre o git-cola para revisão visual
```

Parte 4: Push Protection no GitHub (org)

O que é

Recurso do GitHub que bloqueia `git push` quando detecta segredos conhecidos no conteúdo sendo enviado. Funciona no servidor — não depende de nada instalado na máquina do dev.

Como ativar

1. Acesse: **github.com/organizations/scieloorg/settings/security_analysis**

2. Role até a seção **"Secret scanning"**
3. Clique em **"Enable all"** (ou "Enable for all repositories")
4. Dentro da mesma seção, ative **"Push protection"** — é uma sub-opção que aparece após habilitar Secret scanning
5. Marque também **"Automatically enable for new repositories"**

> Requer permissão de **Organization Owner**.

Como testar

```
# 1. Crie um repositório de teste na org (ou use um existente)
# 2. Adicione um token falso mas com formato reconhecível
echo "ghp_ABCDEFGHIJKLMNOPQRSTUVWXYZabcdef12" > token.txt
git add token.txt
git commit -m "test: token falso"

# 3. Tente fazer push
git push

# Esperado: push REJEITADO pelo GitHub com mensagem explicando
# qual segredo foi detectado e em qual arquivo/linha

# 4. Limpe
git reset --soft HEAD~1
rm token.txt
```

Limitação

O GitHub detecta apenas **padrões de provedores conhecidos** (GitHub tokens, AWS keys, Slack tokens, etc. — são mais de 200 padrões). Ele **não detecta** senhas genéricas como `DATABASE_PASSWORD=abc123`. Por isso as camadas anteriores são essenciais.

Script de setup para distribuir à equipe

Para facilitar a adoção, distribua este script. Cada pessoa executa uma vez:

```
#!/bin/bash
#
# setup-git-security.sh
# Configuração de proteção contra vazamento de segredos.
# Executar UMA VEZ por máquina.
#
set -e

echo "□□Configurando proteção contra vazamento de segredos..."
echo ""

# --- 1. .gitignore global ---
echo "□□[1/4] Configurando .gitignore global..."
mkdir -p ~/.config/git

if ! grep -q "Proteção contra vazamento de segredos" ~/.config/git/ignore 2>/dev/null; then
cat >> ~/.config/git/ignore << 'EOF'

# === Proteção contra vazamento de segredos ===
.envs/
.env
.env.*
.env.local
.env.production
.env.development
.env.staging
*.env
*.pem
*.key
*.p12
*.pfx
credentials.json
service-account*.json
EOF
fi

git config --global core.excludesFile ~/.config/git/ignore
echo "   □ .gitignore global configurado"

# --- 2. Verificar gitleaks ---
```

```
echo ""
echo "□□[2/4] Verificando gitleaks..."
if ! command -v gitleaks &> /dev/null; then
    echo "   ⚠□  gitleaks não encontrado."
    echo "   Instale manualmente:"
    echo "     Ubuntu: wget + dpkg (ver guia)"
    echo "     macOS:  brew install gitleaks"
    echo "   Depois rode este script novamente."
    HAS_GITLEAKS=false
else
    echo "   □ gitleaks $(gitleaks version) encontrado"
    HAS_GITLEAKS=true
fi

# --- 3. Hook pre-commit global ---
echo ""
echo "□□[3/4] Configurando hook pre-commit global..."
mkdir -p ~/.git-hooks

cat > ~/.git-hooks/pre-commit << 'HOOK'
#!/bin/bash
if command -v gitleaks &> /dev/null; then
    CONFIG_FLAG=""
    [ -f "$HOME/.gitleaks.toml" ] && CONFIG_FLAG="--config $HOME/.gitleaks.toml"
    gitleaks git --pre-commit --staged --verbose $CONFIG_FLAG
    if [ $? -ne 0 ]; then
        echo ""
        echo "□ COMMIT BLOQUEADO: segredos detectados!"
        echo "   Remova senhas/tokens e tente novamente."
        echo "   Detalhes: gitleaks git --pre-commit --staged --verbose"
        exit 1
    fi
fi

LOCAL_HOOK=".git/hooks/pre-commit"
[ -f "$LOCAL_HOOK" ] && [ -x "$LOCAL_HOOK" ] && exec "$LOCAL_HOOK"
HUSKY_HOOK="node_modules/.husky/pre-commit"
[ -f "$HUSKY_HOOK" ] && [ -x "$HUSKY_HOOK" ] && exec "$HUSKY_HOOK"
HOOK
```

```
chmod +x ~/.git-hooks/pre-commit
git config --global core.hooksPath ~/.git-hooks
echo "   □ Hook pre-commit global configurado"

# --- 4. git-cola ---
echo ""
echo "□□□ [4/4] Verificando git-cola..."
if command -v git-cola &> /dev/null; then
    echo "   □ git-cola encontrado"
else
    echo "   ⚠□  git-cola não encontrado."
    echo "   Instale: sudo apt install git-cola / brew install git-cola / pip install git-
cola"
fi

git config --global alias.review '!git-cola'
echo "   □ Alias 'git review' configurado"

# --- 5. Config gitleaks customizada ---
if [ "$HAS_GITLEAKS" = true ]; then
    echo ""
    echo "□□Criando configuração customizada gitleaks..."
    cat > ~/.gitleaks.toml << 'EOF'
[extend]
useDefault = true

[[rules]]
id = "env-generic-secret"
description = "Valor suspeito em variável sensível"
regex =
'''(?i)(password|passwd|pwd|secret|token|api_key|apikey|private_key|auth|credential)\s*=\s*['"
]?(\S{8,})['"]?'''
path = '''(\.env|\.env\..+)$'''

[[rules]]
id = "database-url-with-password"
description = "URL de conexão com senha embutida"
regex = '''(postgres|postgresql|mysql|mongodb|redis|amqp)://[^:]+:[^@]{3,}@'''

[allowlist]
```

```
paths = [
    '''\.env\.example$''',
    '''\.env\.sample$''',
    '''\.env\.template$''',
    '''\.env\.dist$''',
]
EOF
    echo "   ~/.gitleaks.toml criado"
fi

echo ""
echo "═══════════════════════════════════════════════"
echo "  Configuração concluída!"
echo ""
echo "  O que foi feito:"
echo "  • .gitignore global → ignora .env, .pem, .key, etc."
echo "  • Hook pre-commit global → gitleaks escaneia todo commit"
echo "  • Regras customizadas → detecta senhas em .env"
echo "  • git-cola → alias 'git review' para revisão visual"
echo ""
echo "  Funciona em TODOS os repos desta máquina."
echo "  Nenhuma configuração por repositório é necessária."
echo "═══════════════════════════════════════════════"
```

Resumo: o que cada camada cobre

Cenário	.gitignore global	gitleaks (hook)	git-cola (visual)	Push Protection (GitHub)
`git add .env`	Bloqueia	—	Mostra no diff	—
`git add --force .env` com senha	Não atua	Bloqueia	Mostra no diff	Se padrão conhecido
Senha hardcoded em `settings.py`	Não atua	Bloqueia	Mostra no diff	Se padrão conhecido
Token AWS em qualquer arquivo	Não atua	Bloqueia	Mostra no diff	Bloqueia
`DATABASE_PASSWORD=abc123` em `.py`	Não atua	Bloqueia	Mostra no diff	Não detecta (genérico)

Cenário	.gitignore global	gitleaks (hook)	git-cola (visual)	Push Protection (GitHub)
`.env` commitado antes da configuração	Não atua	Não atua	Não atua	Não atua

> **Para segredos que já estão no histórico:** é necessário reescrever o histórico com `git filter-repo` ou `BFG Repo-Cleaner` e rotacionar (trocar) todas as credenciais expostas.

Revision #4
Created 12 June 2026 15:10:51 by Jamil Atta
Updated 12 June 2026 17:11:49 by Jamil Atta