

Guia Completo de Git e GitHub no VSCode

Índice

1. [Conceitos Básicos](#)
 2. [Configuração Inicial](#)
 3. [Comandos Básicos](#)
 4. [Trabalhando com Branches](#)
 5. [Comandos Intermediários](#)
 6. [Integração com GitHub](#)
 7. [Interface do VSCode](#)
-

Conceitos Básicos

Git é um sistema de controle de versão distribuído que permite rastrear mudanças no código.

GitHub é uma plataforma online para hospedar repositórios Git e colaborar com outros desenvolvedores.

VSCode possui integração nativa com Git, facilitando o uso através de interface gráfica e terminal integrado.

Configuração Inicial

Instalando Git

Primeiro, certifique-se de que o Git está instalado:

```
bash

# Verificar versão do Git instalada
git --version
```

Configurando Identidade

```
bash
```

```
# Definir seu nome (aparecerá nos commits)
```

```
git config --global user.name "Seu Nome"
```

```
# Definir seu email (use o mesmo do GitHub)
```

```
git config --global user.email "seu.email@exemplo.com"
```

```
# Verificar configurações
```

```
git config --list
```

Configurando Editor Padrão

```
bash
```

```
# Definir VSCode como editor padrão do Git
```

```
git config --global core.editor "code --wait"
```

Comandos Básicos

Inicializando um Repositório

```
bash
```

```
# Criar uma nova pasta para o projeto
```

```
mkdir meu-projeto
```

```
cd meu-projeto
```

```
# Inicializar repositório Git (cria pasta oculta .git)
```

```
git init
```

Status e Verificação

```
bash
```

```
# Ver status dos arquivos (modificados, preparados, não rastreados)
```

```
git status
```

```
# Versão resumida do status
```

```
git status -s
```

Adicionando Arquivos (Staging)

```
bash
```

Adicionar um arquivo específico à área de preparação (staging)

`git add` arquivo.txt

Adicionar vários arquivos específicos

`git add` arquivo1.txt arquivo2.txt

Adicionar todos os arquivos modificados e novos

`git add` .

Adicionar todos arquivos de uma extensão específica

`git add` *.js

Adicionar arquivos de uma pasta específica

`git add` src/

Fazendo Commits

bash

Commit com mensagem descritiva

`git commit -m` "Adiciona funcionalidade de login"

Commit adicionando todos os arquivos modificados automaticamente

(não inclui arquivos novos não rastreados)

`git commit -am` "Atualiza validação de formulário"

Abrir editor para escrever mensagem de commit mais detalhada

`git commit`

Visualizando Histórico

bash

Ver histórico completo de commits

`git log`

Ver histórico resumido (uma linha por commit)

`git log --oneline`

Ver últimos 5 commits

`git log -5`

Ver histórico com gráfico de branches

`git log --oneline --graph --all`

Ver alterações de cada commit

`git log -p`

Ver commits de um autor específico

`git log --author="Nome do Autor"`

Verificando Diferenças

bash

Ver diferenças de arquivos não preparados (working directory vs staging)

`git diff`

Ver diferenças de arquivos preparados (staging vs último commit)

`git diff --staged`

Ver diferenças de um arquivo específico

`git diff arquivo.txt`

Ver diferenças entre dois commits

`git diff commit1 commit2`

Trabalhando com Branches

Conceito de Branches

Branches (ramificações) permitem desenvolver funcionalidades isoladamente sem afetar o código principal.

Comandos de Branch

bash

Listar todas as branches locais

`git branch`

Listar todas as branches (locais e remotas)

`git branch -a`

Criar uma nova branch

`git branch nova-funcionalidade`

Criar e mudar para a nova branch em um comando

`git checkout -b nova-funcionalidade`

Alternativa moderna para criar e mudar de branch

`git switch -c nova-funcionalidade`

Mudar para uma branch existente

`git checkout main`

ou

`git switch main`

Renomear branch atual

`git branch -m novo-nome`

Deletar uma branch (apenas se já foi mesclada)

`git branch -d nome-branch`

Forçar deleção de branch (mesmo não mesclada)

`git branch -D nome-branch`

Mesclando Branches (Merge)

bash

Primeiro, mude para a branch que receberá as alterações

`git checkout main`

Depois, mescle a branch desejada

`git merge nova-funcionalidade`

Se houver conflitos, você precisará resolvê-los manualmente

Após resolver, adicione os arquivos e faça commit

`git add .`

`git commit -m "Resolve conflitos de merge"`

Comandos Intermediários

Desfazendo Alterações

```
bash

# Descartar alterações de um arquivo não preparado
git checkout -- arquivo.txt
# ou (comando moderno)
git restore arquivo.txt

# Remover arquivo da área de preparação (unstage)
git reset HEAD arquivo.txt
# ou (comando moderno)
git restore --staged arquivo.txt

# Desfazer último commit mantendo as alterações nos arquivos
git reset --soft HEAD~1

# Desfazer último commit e alterações na staging area
# (mantém alterações nos arquivos)
git reset HEAD~1

# Desfazer último commit e DESCARTAR todas as alterações
# ⚠ CUIDADO: isso apaga as mudanças permanentemente
git reset --hard HEAD~1
```

Corrigindo Commits

```
bash

# Adicionar alterações ao último commit (sem criar novo commit)
git add arquivo-esquecido.txt
git commit --amend

# Alterar mensagem do último commit
git commit --amend -m "Nova mensagem corrigida"
```

Stash (Guardando Alterações Temporariamente)

```
bash
```

Guardar alterações atuais sem fazer commit
Útil quando precisa mudar de branch mas não quer commitar

`git stash`

Guardar com mensagem descritiva

`git stash save "WIP: trabalhando na função X"`

Listar todas as stashes salvas

`git stash list`

Aplicar a stash mais recente e removê-la da lista

`git stash pop`

Aplicar stash específica (mantém na lista)

`git stash apply stash@{0}`

Ver conteúdo de uma stash

`git stash show -p stash@{0}`

Deletar stash específica

`git stash drop stash@{0}`

Limpar todas as stashes

`git stash clear`

Tags (Marcando Versões)

bash

Criar tag leve (apenas um ponteiro para commit)

```
git tag v1.0.0
```

Criar tag anotada (com mensagem e informações)

```
git tag -a v1.0.0 -m "Versão 1.0.0 - Lançamento inicial"
```

Listar todas as tags

```
git tag
```

Ver informações de uma tag

```
git show v1.0.0
```

Criar tag em commit específico

```
git tag -a v0.9.0 abc123 -m "Versão beta"
```

Deletar tag local

```
git tag -d v1.0.0
```

Rebase (Reorganizando Histórico)

```
bash
```

Aplicar commits da branch atual sobre outra branch

Primeiro, vá para sua branch de trabalho

```
git checkout minha-branch
```

Faça rebase sobre a main

```
git rebase main
```

Se houver conflitos, resolva-os e continue

```
git add .
```

```
git rebase --continue
```

Cancelar rebase

```
git rebase --abort
```

Rebase interativo (reorganizar, editar, unir commits)

Reorganizar últimos 3 commits

```
git rebase -i HEAD~3
```


Integração com GitHub

Conectando com Repositório Remoto

```
bash

# Adicionar repositório remoto
git remote add origin https://github.com/usuario/repositorio.git

# Verificar repositórios remotos configurados
git remote -v

# Alterar URL do repositório remoto
git remote set-url origin https://github.com/usuario/novo-repositorio.git

# Remover repositório remoto
git remote remove origin
```

Enviando Código (Push)

```
bash

# Enviar commits da branch atual para o remoto
git push origin main

# Enviar e definir branch remota como padrão (primeira vez)
git push -u origin main

# Enviar todas as branches
git push --all

# Enviar tags
git push --tags

# Forçar push ( ⚠️ cuidado, reescreve histórico remoto)
git push --force
```

Baixando Código (Pull e Fetch)

```
bash
```

Baixar e mesclar alterações do remoto

`git pull origin main`

Baixar alterações sem mesclar (apenas atualiza referências)

`git fetch origin`

Baixar todas as branches do remoto

`git fetch --all`

Baixar e fazer rebase em vez de merge

`git pull --rebase origin main`

Clonando Repositórios

bash

Clonar repositório completo

`git clone https://github.com/usuario/repositorio.git`

Clonar em pasta com nome específico

`git clone https://github.com/usuario/repositorio.git minha-pasta`

Clonar apenas uma branch específica

`git clone -b nome-branch --single-branch https://github.com/usuario/repositorio.git`

Trabalhando com Pull Requests

bash

1. Criar branch para nova funcionalidade

`git checkout -b feature/nova-funcionalidade`

2. Fazer alterações e commits

`git add .`

`git commit -m "Implementa nova funcionalidade"`

3. Enviar branch para GitHub

`git push -u origin feature/nova-funcionalidade`

4. No GitHub, abrir Pull Request da interface web

5. Após aprovação, atualizar sua branch main local

`git checkout main`

`git pull origin main`

6. Deletar branch local após merge

`git branch -d feature/nova-funcionalidade`

7. Deletar branch remota

`git push origin --delete feature/nova-funcionalidade`

Interface do VSCode

Usando o Painel Source Control

O VSCode possui integração nativa com Git. Acesse através do ícone de controle de versão na barra lateral (terceiro ícone) ou `Ctrl+Shift+G`.

Recursos principais:

- Ver arquivos modificados
- Fazer staging/unstaging clicando no `+` ou `-`
- Escrever mensagem de commit e clicar em `✓` para commitar
- Sincronizar com repositório remoto (push/pull)

Comandos do VSCode

Pressione `Ctrl+Shift+P` (ou `Cmd+Shift+P` no Mac) para abrir a paleta de comandos e digite:

- `Git: Clone` - Clonar repositório
- `Git: Initialize Repository` - Inicializar Git
- `Git: Create Branch` - Criar nova branch

- `Git: Checkout to` - Mudar de branch
- `Git: Commit` - Fazer commit
- `Git: Push` - Enviar alterações
- `Git: Pull` - Baixar alterações
- `Git: Sync` - Sincronizar (pull + push)

Visualizando Diferenças

- Clique em arquivo modificado no painel Source Control para ver diferenças
- Use `Ctrl+Shift+G G` para abrir painel Git
- Extensões recomendadas: GitLens, Git Graph

Terminal Integrado

Use `Ctrl+'` para abrir o terminal integrado do VSCode e executar comandos Git diretamente.

Arquivo .gitignore

Crie um arquivo `.gitignore` na raiz do projeto para ignorar arquivos que não devem ser versionados:

```
bash
```

Dependências

node_modules/

vendor/

Arquivos de ambiente

.env

.env.local

Arquivos de IDE

.vscode/

.idea/

Arquivos do sistema

.DS_Store

Thumbs.db

Logs

*.log

logs/

Arquivos compilados

dist/

build/

*.class

Arquivos temporários

tmp/

temp/

Boas Práticas

Mensagens de Commit

✓ Boas mensagens:

- Adiciona validação de email no formulário de cadastro
- Corrige bug de scroll infinito na página inicial
- Refatora componente Header para usar hooks

✗ Más mensagens:

- mudanças
- fix
- updates

Fluxo de Trabalho Recomendado

1. Sempre puxe alterações antes de começar: `git pull`
2. Crie uma branch para cada funcionalidade: `git checkout -b feature/nome`
3. Faça commits pequenos e frequentes
4. Escreva mensagens descritivas
5. Teste antes de fazer push
6. Use Pull Requests para revisão de código

Resolução de Conflitos

Quando ocorre um conflito durante merge:

1. Abra o arquivo conflitante no VSCode
 2. Você verá marcações: `<<<<<<< HEAD`, `=====`, `>>>>>>> branch-name`
 3. Escolha qual versão manter (ou combine ambas)
 4. Remova as marcações de conflito
 5. Salve o arquivo
 6. Adicione à staging: `git add arquivo.txt`
 7. Complete o merge: `git commit`
-

Comandos Rápidos de Referência

bash

Básicos

```
git init          # Inicializar repositório
git status        # Ver status
git add .         # Adicionar todos arquivos
git commit -m "msg" # Fazer commit
git log --oneline # Ver histórico resumido
```

Branches

```
git branch        # Listar branches
git checkout -b nova # Criar e mudar para branch
git merge outra    # Mesclar branch
```

Remoto

```
git clone url      # Clonar repositório
git remote add origin url # Adicionar remoto
git push origin main # Enviar para remoto
git pull origin main # Baixar do remoto
```

Desfazer

```
git restore arquivo # Descartar alterações
git reset --soft HEAD~1 # Desfazer último commit
```

Stash

```
git stash        # Guardar alterações
git stash pop     # Recuperar alterações
```

Recursos Adicionais

- **Documentação oficial Git:** <https://git-scm.com/doc>
- **GitHub Docs:** <https://docs.github.com>
- **Visualizador interativo:** <https://git-school.github.io/visualizing-git/>
- **Extensão GitLens para VSCode:** Oferece recursos avançados de visualização

💡 **Dica:** Pratique esses comandos em um repositório de teste antes de usar em projetos reais!

🎯 **Próximos passos:** Explore Git Hooks, GitHub Actions e estratégias avançadas de branching como Git Flow.