Guia de Estudo Prático de Comandos Git

Este guia de estudo foi elaborado a partir de um roteiro prático de comandos Git, cobrindo desde a configuração inicial até operações avançadas de *branching* e *merging*. O objetivo é fornecer uma referência estruturada para o aprendizado e a prática do sistema de controle de versão distribuído Git.

---

1. Configuração Inicial e Preparação do Ambiente

Antes de iniciar qualquer projeto, é fundamental configurar a identidade do usuário no Git e preparar o diretório de trabalho.

Ação	Comando Git	Descrição
Definir Nome/Autor	`git config --global user.name "Seu Nome"`	Define o nome do autor que será associado aos seus *commits*.
Definir E-mail	`git config --global user.email "seu.email@exemplo.com"`	Define o e-mail do autor. Recomenda-se usar o mesmo e-mail do GitHub.
Listar Configurações	`git config --list`	Exibe todas as configurações atuais do Git.
Criar Diretório	`mkdir projeto` seguido de `cd projeto`	Cria e navega para o diretório do novo projeto.
Inicializar Repositório	`git init`	Transforma o diretório atual em um repositório Git, criando a pasta oculta `.git`.
Listar Arquivos Ocultos	`ls -a`	Confirma a criação da pasta `.git` (no Linux/macOS).

---

2. O Ciclo de Vida do Commit

O fluxo de trabalho básico no Git envolve a **Working Area** (área de trabalho), a **Staging Area** (área de *staging* ou índice) e o **Repositório** (onde os *commits* são armazenados).

2.1. Adicionar e Comitar Alterações

Ação	Comando Git	Descrição
Adicionar Arquivo Específico	`git add index.html`	Move o arquivo `index.html` da **Working Area** para a **Staging Area**.
Adicionar Todas as Alterações	`git add .`	Move todas as alterações e novos arquivos da **Working Area** para a **Staging Area**.
Realizar Commit	`git commit -m "feat: Mensagem descritiva"`	Cria um *snapshot* permanente das alterações que estão na **Staging Area** e o armazena no repositório.
Commit Rápido (Comitar tudo)	`git commit -a -m "feat: Mensagem"`	Adiciona e comita todas as alterações de arquivos **já rastreados** em uma única etapa.

2.2. Verificação de Status e Histórico

Ação	Comando Git	Descrição
Verificar Status	`git status`	Exibe o estado da **Working Area** e da **Staging Area**, mostrando quais arquivos foram modificados, adicionados ou estão prontos para o *commit*.
Visualizar Histórico Detalhado	`git log`	Exibe o histórico completo de *commits* com detalhes (autor, data, mensagem).
Visualizar Histórico Enxuto	`git log --oneline`	Exibe o histórico de *commits* de forma resumida, mostrando apenas o *hash* curto e a mensagem.
Listar Branches e Heads	`git log --decorate --all`	Exibe o histórico decorado, mostrando onde estão as referências de *branch* e *tags*.

---

3. Inspeção e Desfazimento de Alterações

O Git oferece ferramentas poderosas para inspecionar o que mudou e para desfazer alterações antes que sejam comitadas.

3.1. Comparando Diferenças (`git diff`)

Ação	Comando Git	Descrição
Ver Diferenças (Working vs. Staging)	`git diff`	Mostra as alterações na **Working Area** que ainda não foram adicionadas à **Staging Area**.
Ver Diferenças (Staging vs. Repositório)	`git diff --staged`	Mostra as alterações na **Staging Area** que serão incluídas no próximo *commit*.
Comparar Commits	`git diff HEAD~1 HEAD` ou `git diff hash1 hash2`	Compara o conteúdo entre dois *commits* específicos (por exemplo, o penúltimo e o último *commit*).

3.2. Restaurando e Removendo Alterações

Ação	Comando Git	Descrição
Descartar Alterações (Working Area)	`git restore index.html`	Descarta as alterações feitas no arquivo `index.html` na **Working Area**, restaurando-o para o estado do último *commit* ou da **Staging Area**.
Remover do Staging Area	`git restore --staged index.html`	Remove o arquivo `index.html` da **Staging Area**, movendo-o de volta para a **Working Area** (as alterações no arquivo são mantidas).

Retirar do Staging Area (Alternativa)	`git reset index.html`	Retira o arquivo `index.html` da **Staging Area** (comando mais antigo, mas ainda funcional).

---

4. Gerenciamento de Branches

Branches são fundamentais para o desenvolvimento paralelo, permitindo que diferentes funcionalidades ou correções sejam desenvolvidas isoladamente.

4.1. Visualização e Criação

Ação	Comando Git	Descrição
Verificar Branch Atual	`git branch` ou `git branch --show-current`	Lista todas as *branches* locais e destaca a *branch* atual.
Criar e Mudar para Nova Branch (Antigo)	`git checkout -b form-contato`	Cria a *branch* `form-contato` e muda imediatamente para ela.
Criar e Mudar para Nova Branch (Moderno)	`git switch -c frm-contato`	Método moderno e recomendado para criar e mudar para uma nova *branch*.
Voltar para uma Branch Existente	`git checkout main`	Muda o contexto de trabalho para a *branch* `main`.

4.2. Mesclagem e Exclusão

Ação	Comando Git	Descrição
Mesclar Branches	`git merge frm-contato`	Incorpora as alterações da *branch* `frm-contato` na *branch* atual (neste caso, `main`).
Resolver Conflito	(Manualmente, geralmente via UI do VSCode)	Ocorre quando o Git não consegue mesclar automaticamente as alterações. Requer intervenção manual para

		escolher qual alteração manter.
Excluir Branch Local	`git branch -d form-contato`	Exclui a *branch* local, mas **somente** se ela já tiver sido mesclada.
Excluir Branch Forçadamente	`git branch -D fix-Index`	Exclui a *branch* local, ignorando pendências de mesclagem (útil para *branches* de correção que não serão mescladas).
Trazer Arquivo de Outra Branch	`git checkout fix-Index -- style.css`	Traz o arquivo `style.css` da *branch* `fix-Index` para a *branch* atual (útil para mover arquivos específicos).

---

5. Fluxo de Trabalho Prático

O roteiro original sugere um fluxo de trabalho prático que pode ser resumido nas seguintes etapas:

1. **Configurar** o Git (`git config`).
2. **Inicializar** o repositório (`git init`).
3. **Criar** e **Modificar** arquivos.
4. **Verificar** o estado (`git status`).
5. **Adicionar** as alterações (`git add .`).
6. **Comitar** as alterações (`git commit -m "..."`).
7. **Criar** uma *branch* para uma nova funcionalidade ou correção (`git switch -c nova-feature`).
8. **Desenvolver** e **Comitar** na nova *branch*.
9. **Voltar** para a *branch* principal (`git checkout main`).
10. **Mesclar** as alterações (`git merge nova-feature`).
11. **Excluir** a *branch* de trabalho (`git branch -d nova-feature`).

Este guia de estudo cobre os comandos essenciais para a gestão de projetos com Git, conforme detalhado no arquivo de origem. Recomenda-se a prática constante desses comandos para a consolidação do aprendizado.