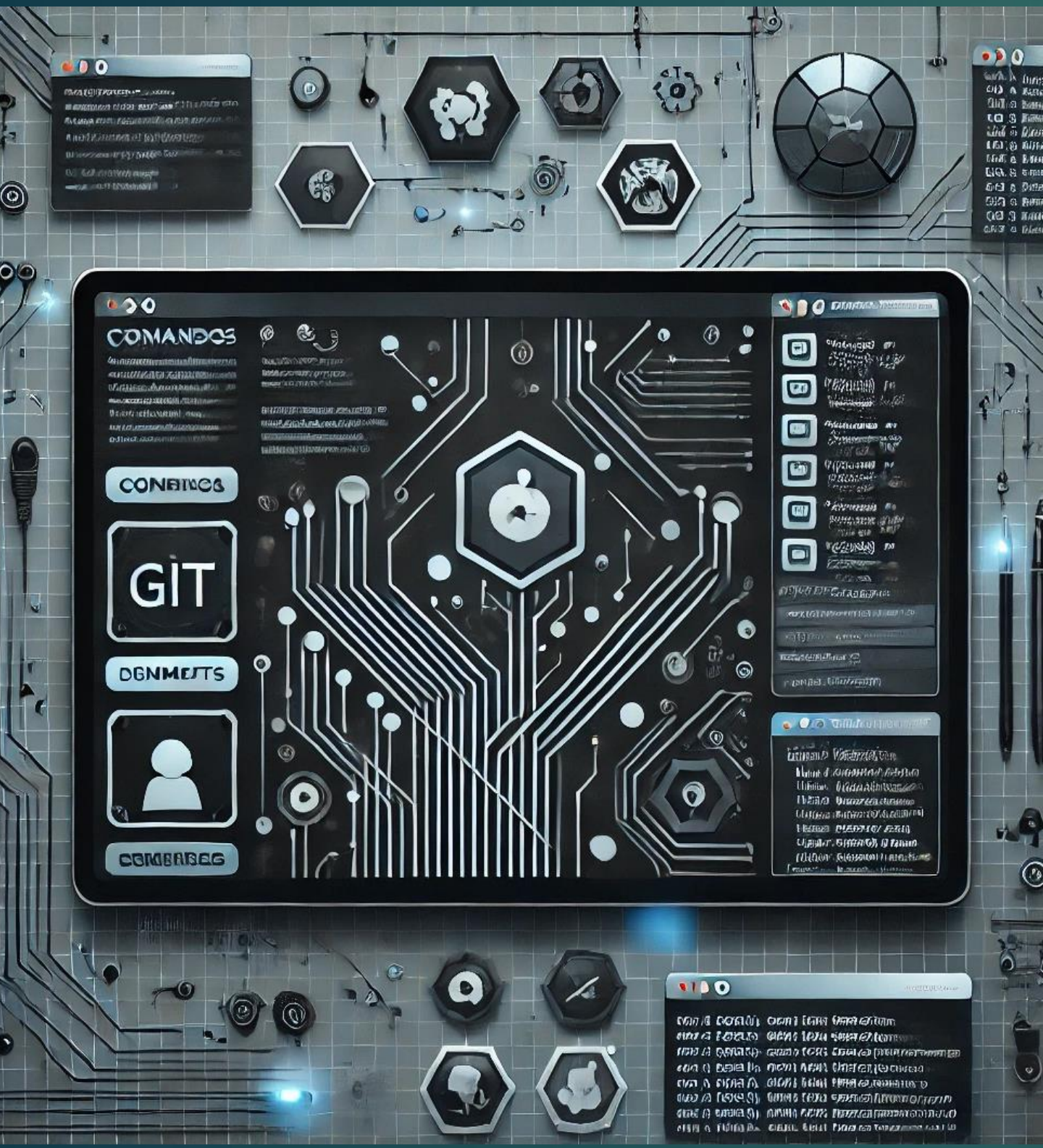
Dominando o Git: Do Básico ao Avançado



Maria C. Adriano

Introdução ao Git

O Git é um sistema de controle de versão distribuído amplamente utilizado por desenvolvedores em todo o mundo. Criado por Linus Torvalds em 2005, ele permite que equipes colaborem de maneira eficiente no desenvolvimento de software, mantendo um histórico detalhado de todas as modificações feitas nos arquivos de um projeto.

O que é Git e para que serve?

O Git é essencial para o desenvolvimento moderno de software, pois:

- Permite rastrear alterações no código-fonte.
- Facilita a colaboração entre múltiplos desenvolvedores.
- Mantém um histórico seguro de todas as modificações.
- Possibilita a reversão de alterações em caso de erro.

01

Instalação e Configuração

Instalação e Configuração

Instalando o Git

Para instalar o Git, siga os passos abaixo conforme seu sistema operacional:

Windows:

1. Acesse git-scm.com e baixe o instalador.
2. Execute o instalador e siga as instruções na tela.
3. Verifique a instalação abrindo o terminal (CMD ou PowerShell) e digitando:

```
git –version
```

Linux (Ubuntu/Debian):

```
sudo apt update
sudo apt install git
```

MacOS:

```
brew install git
```

Configurando o Git

Após instalar o Git, configure seu nome de usuário e e-mail:

```
git config --global user.name "Seu Nome“
git config --global user.email seu@email.com
```

Para verificar as configurações:

```
git config –list
```

Diferença entre Git e GitHub

Muitas pessoas confundem Git e GitHub, mas são conceitos diferentes:Git é um sistema de controle de versão.GitHub é um serviço online para armazenar repositórios Git, facilitando a colaboração remota.

02

Comandos Básicos

Comandos Básicos do Git

Agora que você instalou e configurou o Git, vamos explorar os comandos fundamentais para começar a trabalhar com ele.

Criando e Inicializando um Repositório

Para iniciar um novo repositório Git em um diretório, utilize:

```
git init
```

Isso criará um novo repositório Git local.

Se você deseja clonar um repositório remoto, utilize:

```
git clone <URL_DO_REPOSITORIO>
```

Isso copiará todos os arquivos e o histórico de versão para o seu computador.

Verificando o Status dos Arquivos

O comando abaixo exibe o status atual dos arquivos no repositório:

```
git status
```

Ele mostra quais arquivos foram modificados, adicionados ou removidos do repositório.

Adicionando Arquivos ao Controle de Versão

Para adicionar arquivos ao controle de versão, use:

```
git add <nome_do_arquivo>
```

Para adicionar todos os arquivos modificados de uma vez:

```
git add .
```

Criando um Commit

Um commit é um ponto salvo no histórico do repositório. Para criar um commit, use:

```
git commit -m "Mensagem descritiva do commit“
```

Certifique-se de escrever mensagens de commit claras e descritivas.

Visualizando o Histórico de Commits.

Para visualizar os commits feitos no repositório, utilize:

```
git log
```

Isso exibirá uma lista com os commits, autores e datas.

03

Trabalhando com Branches

Trabalhando com Branches no Git

Branches (ou ramificações) permitem trabalhar em diferentes funcionalidades ou correções sem afetar a branch principal do projeto. Vamos explorar como criar, listar, alternar e mesclar branches no Git.

Criando uma Nova Branch

Para criar uma nova branch, utilize:

```
git branch <nome-da-branch>
```

Isso cria uma nova branch, mas não muda para ela automaticamente.

Alternando entre Branches

Para mudar para uma branch existente, utilize:

```
git checkout <nome-da-branch>
```

Ou, a partir do Git 2.23, use:

```
git switch <nome-da-branch>
```

Listando Branches

Para ver todas as branches disponíveis no repositório, use:

```
git branch
```

A branch atual será indicada com um *.

Mesclando Branches

Após finalizar o trabalho em uma branch, podemos mesclá-la à branch principal (main ou master):

```
git checkout main  # Certifique-se de estar na branch principal
git merge <nome-da-branch>
```

Isso incorporará as mudanças da branch especificada na branch atual.

Deletando Branches

Após a mesclagem, você pode excluir a branch desnecessária:

```
git branch -d <nome-da-branch>
```

Se a branch ainda não tiver sido mesclada e quiser forçar a exclusão:

```
git branch -D <nome-da-branch>
```

04

Trabalho Remoto com GitHub/GitLab

Trabalhando com Repositórios Remotos e Colaboração no GitHub

O Git permite que você trabalhe com repositórios remotos, o que é essencial para a colaboração em projetos. O GitHub, GitLab e Bitbucket são plataformas populares que hospedam repositórios Git, permitindo que desenvolvedores contribuam com código e compartilhem seus projetos.

Conectando-se a um Repositório Remoto

Para associar seu repositório local a um repositório remoto no GitHub.

```
git remote add origin <URL_DO_REPOSITORIO>
```

Para verificar os repositórios remotos configurados:

```
git remote –v
```

Enviando Alterações para o Repositório Remoto

Depois de realizar commits no repositório local, você pode enviá-los para o repositório remoto usando:

```
git push origin <nome-da-branch>
```

Se for a primeira vez enviando commits na branch principal:

```
git push -u origin main
```

Obtendo Alterações do Repositório Remoto

Para atualizar seu repositório local com as alterações do repositório remoto, use:

```
git pull origin <nome-da-branch>
```

Clonando um Repositório

Caso você queira baixar um repositório remoto para seu computador, utilize:

```
git clone <URL_DO_REPOSITORIO>
```

Criando Pull Requests no GitHub

Um Pull Request (PR) é uma solicitação para mesclar alterações em um repositório.

Para criar um PR:

Faça um fork do repositório, caso não tenha permissão para alterar diretamente.Crie uma nova branch e faça suas alterações.
Suba suas alterações para o GitHub com:

```
git push origin <nome-da-branch>.
```

Acesse o repositório no GitHub e clique em "Compare & pull request".Descreva suas mudanças e envie a solicitação.

05

Resolvendo Conflitos

Resolvendo Conflitos no Git

Ao trabalhar com Git, é comum encontrar conflitos ao tentar mesclar branches ou puxar alterações do repositório remoto. Isso acontece quando duas ou mais pessoas modificam a mesma linha de um arquivo em commits diferentes.

Identificando Conflitos

Quando um conflito ocorre, o Git interrompe a mesclagem e exibe uma mensagem de erro, como:

```
CONFLICT (content): Merge conflict in arquivo.txtAutomatic merge failed; fix conflicts and then commit the result.
```

Para verificar os arquivos com conflito, utilize:

```
git status
```

Como Resolver Conflitos

1. Abrir o arquivo com conflito

No arquivo afetado, o Git insere marcações para indicar as diferenças:

```
<<<<<<< HEAD
Esta é a alteração na branch atual.
=======
Esta é a alteração na outra branch.
>>>>>>> nome-da-branch
```

O código entre <<<<<<< HEAD e ======= pertence à branch atual.
O código abaixo de ======= pertence à outra branch.

2. Editar o arquivo para manter apenas a versão desejada

Após revisar as diferenças, remova as marcações <<<<<<<, ======= e >>>>>>> e mantenha apenas o código correto.

3. Adicionar o arquivo ao stage

Depois de resolver o conflito, adicione o arquivo novamente:

```
git add arquivo.txt
```

4. **Criar um commit para concluir a resolução**

```
git commit -m "Resolvendo conflito no arquivo.txt“
```

Cancelando uma Mesclagem com Conflitos

Se decidir cancelar a mesclagem e voltar ao estado anterior:

```
git merge –abort.
```

06

Trabalhando com Tags e Controle de Versões no Git

Trabalhando com Tags e Controle de Versões no Git

As tags no Git são usadas para marcar pontos importantes no histórico do repositório, como versões de lançamento. Elas são especialmente úteis para indicar versões estáveis do projeto.

Criando e Gerenciando Tags

Existem dois tipos principais de tags no Git:
Tags Anotadas - Contêm metadados, como autor, data e mensagem.
Tags Leves - São apenas ponteiros para um commit específico.

Criando uma Tag Leve

Para criar uma tag leve, basta executar:

```
git tag nome-da-tag
```

Criando uma Tag Anotada

Uma tag anotada inclui informações extras e é criada com:

```
git tag -a v1.0 -m "Versão estável 1.0“
```

Isso cria a tag v1.0 com a mensagem "Versão estável 1.0".

Listando e Visualizando Tags

Para listar todas as tags no repositório:

```
git tag
```

Se quiser visualizar detalhes de uma tag anotada específica:

```
git show v1.0
```

Publicando Tags no Repositório Remoto

Por padrão, as tags não são enviadas automaticamente ao repositório remoto. Para enviá-las, use:

```
git push origin v1.0
```

Para enviar todas as tags de uma vez:

```
git push origin –tags
```

Removendo Tags

Para remover uma tag local:

```
git tag -d v1.0
```

Se precisar remover uma tag do repositório remoto:

```
git push origin --delete v1.0
```

O uso correto de tags facilita o gerenciamento de versões e o rastreamento de lançamentos no Git.

07

Comandos Avançados e Boas Práticas

Comandos Avançados e Boas Práticas no Git

Neste capítulo, você explorará comandos avançados do Git que facilitam a resolução de problemas e a manutenção do histórico de versões, além de conhecer dois fluxos de trabalho comuns utilizados em equipes profissionais.

Comandos Avançados do Git

Git cherry-pick – Aplicando Commits Específicos

O `git cherry-pick` é uma ferramenta poderosa para importar commits isolados de outra branch.

Exemplo prático:

```
git checkout main
git cherry-pick a1b2c3d
```

-Dica: Use `-x` para documentar a origem do commit:

```
git cherry-pick -x a1b2c3d
```

Git bisect` – Encontrando Commits com Bugs

Com o `git bisect`, você pode fazer uma busca binária para localizar qual commit introduziu um erro.

Fluxo básico:

```
git bisect start
git bisect bad
git bisect good <commit-hash>
```

Automatização com scripts:

```
git bisect run ./test-script.sh
```

Git blame – Identificando Autores de Alterações

Use `git blame` para investigar quem modificou cada linha de um arquivo.

Exemplo:

```
git blame arquivo.js | grep 'funcaoX'
```

Fluxos de Trabalho no Git

Git Flow – Organização Estruturada
-Principais Branches:`main`, `develop`, `feature`, `release`,hotfix`.

Exemplos de comandos:

```
git flow init
git flow feature start nova-feature
git flow feature finish nova-feature
```

Trunk-Based Development – Agilidade com Simplicidade

Característica:Desenvolvimento contínuo diretamente na branch principal (`main`).

Boas práticas: Commits pequenos e frequentes, uso de *feature flags* para lançamentos parciais.

Boas Práticas com Git

1. Escreva mensagens de commit claras:Prefira o padrão "tipo: breve descrição".
2. Faça revisões de código: Adote Pull Requests para melhoria contínua.
3. Atualize frequentemente:Use `git fetch` e `git rebase` para manter o histórico organizado.
4. Proteja a branch principal: Ative revisões obrigatórias antes de merges.

08

Comandos Avançados: Rebase e Cherry-Pick

Técnicas Avançadas: Rebase e Cherry-Pick

Além do merge, o Git oferece comandos avançados para manipular commits e organizar o histórico do repositório. Dois dos mais úteis são rebase e cherry-pick.

Git Rebase

O git rebase é usado para reescrever o histórico de commits, movendo uma sequência de commits para uma nova base.

Diferença entre Merge e Rebase

git merge: Junta os commits de outra branch, mantendo o histórico de commits original.
git rebase: Move commits de uma branch para outra, reorganizando o histórico e evitando merges adicionais.

Como Usar o Rebase

Se você está em uma branch secundária e deseja trazer as mudanças da branch main sem criar um commit de merge, use:

```
git checkout minha-branch
git rebase main
```

Se houver conflitos, resolva-os manualmente e continue o rebase com:

```
git add
.git rebase –continue
```

Se precisar cancelar o rebase:

```
git rebase –abort
```

Git Cherry-Pick

O git cherry-pick permite aplicar um commit específico de uma branch em outra, sem precisar mesclar todas as mudanças.

Como Usar o Cherry-Pick
Liste os commits da branch de origem:

```
git log –oneline
```

Pegue o hash do commit desejado (exemplo: a1b2c3d) e aplique-o na branch atual:

```
git cherry-pick a1b2c3d
```

Isso copia apenas aquele commit específico para a branch ativa.Se ocorrer um conflito durante o cherry-pick, resolva-o e finalize com:

```
git cherry-pick –continue
```

Para cancelar o cherry-pick:

```
git cherry-pick –abort
```

Essas técnicas são essenciais para manter um histórico limpo e organizado no Git.

09

Aliases no Git

Aliases no Git – Comandos Personalizados

Crie atalhos para agilizar comandos frequentes:

```
git config --global alias.co checkout
git config --global alias.br branch
git config --global alias.st status
git config --global alias.last 'log -1 HEAD‘
```

Verifique seus aliases: git config --global –list

Comandos para Otimizar Repositórios

Remover branches locais já mescladas:

```
git fetch –p
git branch --merged | grep -v '\*' | xargs -n 1 git branch –d
```

Limpar arquivos não rastreados:

```
git clean -f –d
```

Compactar repositório:

```
git gc --aggressive --prune=now
```

Ferramentas e Cursos Recomendados

Ferramentas:
GitKraken (Interface gráfica intuitiva).
Sourcetree (Gerenciamento visual de branches).
GitHub Desktop (Integração com GitHub).

Cursos:
Git e GitHub para Iniciantes (Udemy).
Versionamento com Git (Alura).
Learn Git Branching (Interativo e gratuito).

Compreender esses comandos, fluxos e práticas ajudará você a dominar o Git, mantendo seu trabalho organizado e eficiente.

Encerramento

Página de Encerramento

Conclusão

Você chegou ao fim deste eBook! Esperamos que os conhecimentos compartilhados sobre comandos avançados do Git sejam valiosos para seu crescimento como desenvolvedor.

Aprofundar-se no Git é mais do que dominar comandos: é compreender processos, colaborar com eficiência e manter projetos organizados. Com as boas práticas apresentadas, você estará preparado para enfrentar desafios reais e trabalhar com equipes de alta performance.

Próximos Passos

1. Pratique diariamente: Experimente os comandos em repositórios pessoais.
2. Colabore em projetos open source: Exercite fluxos como o Git Flow e Trunk-Based.
3. Explore novas ferramentas: Integre suas práticas com interfaces gráficas e automações.

Agradecimento

Obrigado por investir seu tempo neste aprendizado! Que este conteúdo seja um guia em sua jornada tecnológica. Continue explorando, criando e compartilhando conhecimento.

Bons commits e até a próxima versão!