Guia Completo de Comandos Git - Ordenado por Importância

🔥 COMANDOS ESSENCIAIS (Uso Diário)

1. `git status`

O que faz: Mostra o estado atual do repositório

```bash
git status
git status -s  # versão curta
```

2. `git add`

O que faz: Adiciona arquivos ao staging area

```bash
git add arquivo.txt          # adiciona arquivo específico
git add .                    # adiciona todos os arquivos modificados
git add -A                   # adiciona todos (incluindo deletados)
git add -p                   # adiciona interativamente (por pedaços)
```

3. `git commit`

O que faz: Salva as mudanças no repositório

```bash
git commit -m "mensagem"     # commit com mensagem
git commit -am "mensagem"    # add + commit para arquivos já trackeados
git commit --amend           # modifica o último commit
git commit --amend --no-edit # modifica último commit sem alterar mensagem
```

4. `git push`

O que faz: Envia commits para o repositório remoto

```bash
git push                     # push para branch atual
git push origin main         # push para branch específica
git push -u origin feature   # push + define upstream
git push --force-with-lease  # push forçado seguro
git push --tags              # push com tags
```

5. `git pull`

O que faz: Baixa e integra mudanças do repositório remoto

```bash
git pull                    # pull da branch atual
git pull origin main        # pull de branch específica
git pull --rebase           # pull com rebase ao invés de merge
git pull --ff-only          # pull apenas se for fast-forward
```

6. `git clone`

O que faz: Clona um repositório remoto

```bash
git clone https://github.com/user/repo.git
git clone --depth 1 url     # clone shallow (apenas último commit)
git clone -b branch url     # clone de branch específica
```

7. `git branch`

O que faz: Gerencia branches

```bash
git branch                  # lista branches locais
git branch -a               # lista todas as branches
git branch nova-feature     # cria nova branch
git branch -d feature       # deleta branch (safe)
git branch -D feature       # deleta branch (force)
git branch -m novo-nome     # renomeia branch atual
```

8. `git checkout` / `git switch`

O que faz: Muda entre branches ou restaura arquivos

```bash
# Mudança de branch (ambos funcionam)
git checkout main
git switch main

# Criar e mudar para nova branch
git checkout -b nova-feature
git switch -c nova-feature

# Restaurar arquivo
git checkout -- arquivo.txt
git restore arquivo.txt     # comando mais novo
```

9. `git merge`

O que faz: Integra mudanças de uma branch em outra

```bash
git merge feature-branch    # merge simples
git merge --no-ff feature   # merge sem fast-forward
git merge --squash feature  # merge com squash
```

10. `git log`

O que faz: Mostra histórico de commits

```bash
git log                     # log completo
git log --oneline           # log resumido
git log --graph             # log com gráfico
git log -p                  # log com diff
git log --since="2 weeks ago"
git log --author="nome"
```

🚀 COMANDOS INTERMEDIÁRIOS (Uso Frequente)

11. `git diff`

O que faz: Mostra diferenças entre commits, branches ou arquivos

```bash
git diff                    # diferenças não staged
git diff --staged           # diferenças staged
git diff HEAD~1             # diferenças do último commit
git diff branch1..branch2   # diferenças entre branches
```

12. `git stash`

O que faz: Salva mudanças temporariamente

```bash
git stash                   # salva mudanças atuais
git stash push -m "msg"     # stash com mensagem
git stash list              # lista stashes
git stash pop               # aplica e remove último stash
git stash apply             # aplica sem remover
git stash drop              # remove stash
```

13. `git remote`

O que faz: Gerencia repositórios remotos

```bash
git remote -v              # lista remotos
git remote add origin url  # adiciona remoto
git remote set-url origin nova-url
git remote remove origin  # remove remoto
```

14. `git fetch`

O que faz: Baixa mudanças sem integrar

```bash
git fetch                  # fetch de todos os remotos
git fetch origin           # fetch de remoto específico
git fetch --prune          # remove referências mortas
```

15. `git reset`

O que faz: Desfaz commits ou mudanças

```bash
git reset HEAD~1           # desfaz último commit (soft)
git reset --soft HEAD~1    # desfaz commit, mantém mudanças staged
git reset --hard HEAD~1    # desfaz commit e mudanças (CUIDADO!)
git reset arquivo.txt      # remove do staging
```

16. `git revert`

O que faz: Reverte commits de forma segura

```bash
git revert HEAD            # reverte último commit
git revert commit-hash     # reverte commit específico
git revert --no-commit HEAD~2..HEAD  # reverte range sem commit
```

17. `git tag`

O que faz: Gerencia tags (versões)

bash

```bash
git tag                    # lista tags
git tag v1.0.0             # cria tag simples
git tag -a v1.0.0 -m "Release 1.0"  # tag anotada
git push origin --tags     # push das tags
```

⚡ COMANDOS AVANÇADOS (Resolução de Problemas)

18. `git rebase`

O que faz: Reaproveita commits em nova base

bash

```bash
git rebase main            # rebase na main
git rebase -i HEAD~3       # rebase interativo (últimos 3)
git rebase --continue      # continua após resolver conflitos
git rebase --abort         # cancela rebase
```

19. `git cherry-pick`

O que faz: Aplica commits específicos de outras branches

bash

```bash
git cherry-pick commit-hash
git cherry-pick -x commit-hash  # mantém referência original
git cherry-pick commit1..commit3  # range de commits
```

20. `git reflog`

O que faz: Mostra histórico de mudanças nas referências (salvação!)

bash

```bash
git reflog                 # histórico completo
git reflog --oneline       # versão resumida
git reset --hard HEAD@{2}  # volta para estado anterior
```

21. `git bisect`

O que faz: Encontra commit que introduziu bug

bash

```bash
git bisect start
git bisect bad           # commit atual é ruim
git bisect good v1.0     # commit bom conhecido
git bisect reset         # termina bisect
```

22. `git clean`

O que faz: Remove arquivos não trackeados

bash

```bash
git clean -n             # mostra o que seria removido
git clean -f             # remove arquivos
git clean -fd            # remove arquivos e diretórios
git clean -fX            # remove apenas ignored files
```

23. `git blame`

O que faz: Mostra quem modificou cada linha

bash

```bash
git blame arquivo.txt
git blame -L 10,20 arquivo.txt  # linhas específicas
```

24. `git show`

O que faz: Mostra informações detalhadas de commits

bash

```bash
git show                 # último commit
git show commit-hash     # commit específico
git show HEAD~2:arquivo.txt  # arquivo em commit específico
```

🔧 COMANDOS DE CONFIGURAÇÃO

25. `git config`

O que faz: Configura Git

bash

```bash
git config --global user.name "Seu Nome"
git config --global user.email "email@example.com"
git config --list        # lista configurações
git config --global init.defaultBranch main
```

26. `git init`

O que faz: Inicializa repositório Git

```bash
git init                 # inicializa repo atual
git init --bare          # repositório bare (servidor)
```

🆘 COMANDOS PARA EMERGÊNCIAS

27. `git fsck`

O que faz: Verifica integridade do repositório

```bash
git fsck                 # verificação básica
git fsck --full          # verificação completa
```

28. `git gc`

O que faz: Limpa e otimiza repositório

```bash
git gc                   # limpeza básica
git gc --aggressive      # limpeza agressiva
git gc --prune=now       # remove objetos órfãos
```

29. `git archive`

O que faz: Cria arquivo compactado do repositório

```bash
git archive --format=zip HEAD > projeto.zip
git archive --format=tar.gz --prefix=projeto/ HEAD > projeto.tar.gz
```

30. `git submodule`

O que faz: Gerencia submódulos

```bash
git submodule add url path
git submodule update --init --recursive
git submodule foreach git pull origin main
```

💡 COMBINAÇÕES ÚTEIS PARA O DIA A DIA

Workflow típico:

```bash
git status                    # verificar estado
git add .                     # adicionar mudanças
git commit -m "feat: nova funcionalidade"
git push                      # enviar para GitHub
```

Sincronizar com remoto:

```bash
git fetch --prune             # atualizar referências
git pull --rebase             # puxar mudanças com rebase
git push                      # enviar mudanças
```

Resolver conflitos:

```bash
git status                    # ver arquivos em conflito
# editar arquivos para resolver conflitos
git add arquivo-resolvido.txt
git commit                    # finalizar merge/rebase
```

Desfazer mudanças perigosas:

```bash
git reflog                    # ver histórico
git reset --hard HEAD@{2}     # voltar para estado anterior
```

🎯 DICAS PRO

1. **Sempre use `--force-with-lease` ao invés de `--force`**
2. **Configure aliases para comandos frequentes**
3. **Use `git stash` antes de mudanças arriscadas**
4. **Mantenha commits pequenos e descritivos**
5. **Use `git reflog` quando algo der errado**
6. **Configure `.gitignore` desde o início**
7. **Use branches para features**
8. **Faça pull requests pequenos e focados**