Manual Unificado de Comandos Git

# 1) Git + SSH + main (Windows/Git Bash)

Este guia cobre os passos iniciais para configurar SSH, conectar repositórios ao GitHub e definir a branch main como padrão.

* ssh-keygen -t ed25519 -C "seu-email@github.com"
* eval "$(ssh-agent -s)"
* ssh-add ~/.ssh/id\_ed25519
* clip < ~/.ssh/id\_ed25519.pub (copiar chave para o GitHub)
* ssh -T git@github.com
* git init, git add ., git commit -m "primeiro commit"
* git remote add origin git@github.com:SEU\_USUARIO/SEU\_REPO.git
* git branch -M main
* git push -u origin main
* git config --global init.defaultBranch main

# 2) Guia Rápido - git add

|  |  |  |
| --- | --- | --- |
| Comando | O que faz | Quando usar |
| git add . | Adiciona todos os arquivos novos e modificados a partir da pasta atual (e subpastas). | Quando você está na raiz do projeto e quer adicionar tudo de uma vez. |
| git add -A | Adiciona todas as mudanças em todo o repositório (novos, modificados e deletados). | Quando quer garantir que inclusive deleções sejam registradas. |
| git add -u | Adiciona apenas arquivos modificados e deletados (não inclui novos). | Quando só alterou/removeu arquivos, mas não criou nada novo. |
| git add <arquivo> | Adiciona um arquivo específico. | Quando quer controlar exatamente qual arquivo vai para o próximo commit. |
| git add <pasta>/ | Adiciona todos os arquivos dentro de uma pasta específica. | Quando quer commitar só uma parte do projeto (ex.: git add docs/). |

# 3) Dicas Extras

Mover vários arquivos para uma pasta:

mv \*.\* Git\_GitHub/

mv \*.pdf Git\_GitHub/

mv arquivo1.pdf arquivo2.docx arquivo3.txt Git\_GitHub/

git add -A && git commit -m "organiza arquivos na pasta Git\_GitHub" && git push origin main

Exemplo de .gitignore para evitar arquivos soltos:

/\*  
!\*/  
!.gitignore

# 4) Comandos git log e git restore

git log → mostra histórico de commits com hash, autor, data e mensagem.

Variações úteis: --oneline, -5 --oneline, --stat, git log NOME\_DO\_ARQUIVO.

git restore → descarta mudanças locais ou remove arquivos do stage.

Exemplos: git restore arquivo, git restore --staged arquivo, git restore .

# 5) .gitignore

O arquivo .gitignore define padrões de arquivos e pastas que o Git deve ignorar, ou seja, não rastrear nem incluir em commits.

Exemplo de conteúdo para .gitignore:

# Ignorar arquivos temporários  
\*.log  
\*.tmp  
  
# Ignorar pastas  
node\_modules/  
\_\_pycache\_\_/

# 6) git tag

Tags são usadas para marcar pontos importantes na história do repositório, geralmente versões.

Principais usos:

|  |  |
| --- | --- |
| Comando | Descrição |
| git tag v1.0 | Cria uma tag leve no commit atual. |
| git tag -a v1.0 -m "Primeira versão estável" | Cria uma tag anotada com mensagem e autor. |
| git tag | Lista todas as tags. |
| git show v1.0 | Mostra detalhes do commit marcado pela tag. |
| git push origin v1.0 | Envia uma tag específica para o remoto. |
| git push origin --tags | Envia todas as tags locais para o remoto. |

# 7) git alias

Aliases permitem criar atalhos personalizados para comandos Git, facilitando o uso diário.

|  |  |
| --- | --- |
| Comando | Descrição |
| git config --global alias.co checkout | Atalho: git co → git checkout. |
| git config --global alias.ci commit | Atalho: git ci → git commit. |
| git config --global alias.st status | Atalho: git st → git status. |
| git config --global alias.lg "log --oneline --graph --all --decorate" | Atalho: git lg → log resumido e gráfico. |

# 8) git rebase — conceito e uso

O rebase reaplica seus commits em cima de outra base (geralmente a main atualizada), mantendo um histórico mais linear (sem commits de merge). É muito útil para atualizar sua branch de trabalho antes de abrir um Pull Request.

Fluxo típico para atualizar sua feature com a main mais recente:

git checkout feature # esteja na branch de trabalho

git fetch origin # busca atualizações do remoto

git rebase origin/main # reaplica seus commits sobre a main atualizada

# se houver conflitos: edite os arquivos, depois:

git add . # marca conflito resolvido

git rebase --continue # continua o rebase até terminar

# se quiser desistir do rebase:

git rebase --abort

git push --force-with-lease # ao final, envie (rebase reescreve histórico)

|  |  |
| --- | --- |
| Comando | Descrição |
| git rebase BRANCH | Reaplica os commits da branch atual em cima de BRANCH. |
| git rebase --continue | Continua o rebase após resolver conflitos. |
| git rebase --abort | Cancela o rebase e volta ao estado anterior. |
| git push --force-with-lease | Envia alterações após rebase de forma segura (protege caso alguém tenha pushado algo). |

# 9) merge x rebase — quando usar

• Use MERGE quando for juntar o trabalho final da feature na main preservando o histórico como aconteceu.

• Use REBASE quando quiser atualizar sua feature com a main mantendo histórico linear (sem commits de merge).

Exemplo de merge (feature → main, estando na main):

git checkout main

git pull origin main

git merge feature

git push origin main

# 10) Resolver conflitos (merge/rebase)

Passos gerais:

git status # veja arquivos em conflito

Abra os arquivos e resolva os marcadores: <<<<<<<, =======, >>>>>>>

git add ARQUIVO(S) # marque como resolvido

git commit # (no merge) finaliza a mesclagem

git rebase --continue # (no rebase) continua o processo

git merge --abort / git rebase --abort # aborta se necessário

|  |  |
| --- | --- |
| Comando | Descrição |
| git checkout --ours CAMINHO/ARQUIVO | Fica com a versão da branch atual. |
| git checkout --theirs CAMINHO/ARQUIVO | Fica com a versão da outra branch (a que está sendo mesclada). |

Dica: Após resolver tudo, rode um teste rápido do projeto antes do push.

# 11) git stash — guardar alterações temporariamente

Use o stash quando você quer trocar de branch ou atualizar a base, mas tem alterações locais que ainda não quer commitar.

|  |  |
| --- | --- |
| Comando | Descrição |
| git stash | Guarda alterações não commitadas e limpa a pasta de trabalho. |
| git stash list | Lista entradas armazenadas no stash. |
| git stash pop | Aplica a última entrada e remove do stash. |
| git stash apply | Aplica a entrada mas mantém no stash. |
| git stash push -m "mensagem" | Guarda alterações com uma mensagem identificadora. |
| git stash drop | Remove a última entrada do stash. |
| git stash clear | Limpa todas as entradas do stash (cuidado!). |

Fluxo típico:

git status # vê o que mudou

git stash # guarda as mudanças

git checkout OUTRA\_BRANCH

git pull / git rebase # faz o que precisa

git checkout sua\_branch

git stash pop # recupera as mudanças para continuar trabalhando

# 12) Fork, upstream e sincronização

Em um fork, 'origin' aponta para o SEU repositório no GitHub. 'upstream' aponta para o repositório original (ex.: o da professora). Para manter seu fork atualizado:

git remote add upstream https://github.com/jaquelinedealmeida/pretalab-git-github-avancado.git # exemplo

git remote -v # confira origin (seu fork) e upstream (repo original)

git fetch upstream # busca atualizações do repo original

git checkout main

git merge upstream/main # ou: git rebase upstream/main

git push origin main # envia sua main atualizada para seu fork

# Alternativa via GitHub UI: botão 'Sync fork' → 'Update branch'

# 13) O que é o -u no git push (upstream)

Ao fazer 'git push -u origin NOME\_DA\_BRANCH', você define a branch remota padrão (upstream). Depois disso, pode usar apenas 'git push' e 'git pull' sem precisar especificar origin/branch.

# 14) Tabela-resumo — comandos que você treinou

|  |  |
| --- | --- |
| Comando | Descrição |
| git init | Inicia um repositório. |
| git status | Mostra estado dos arquivos. |
| git add . / -A / -u | Adiciona mudanças à área de stage (variações). |
| git commit -m "msg" | Registra o snapshot no histórico. |
| git branch / git branch NOME | Lista/cria branches. |
| git checkout -b NOME / git switch -c NOME | Cria e muda para a nova branch. |
| git checkout NOME / git switch NOME | Muda para outra branch existente. |
| git branch -m NOVO | Renomeia a branch atual. |
| git branch -d/-D NOME | Deleta branch local (forçar com -D). |
| git push -u origin NOME | Envia branch e define upstream. |
| git push origin --delete NOME | Deleta branch remota. |
| git fetch --all | Busca referências remotas. |
| git checkout -b NOME origin/NOME | Cria branch local a partir de remota. |
| git merge OUTRA\_BRANCH | Mescla OUTRA\_BRANCH na branch atual. |
| git rebase BRANCH | Reaplica commits sobre BRANCH (histórico linear). |
| git log / git log --oneline | Histórico completo/resumido. |
| git restore ARQ / --staged | Descarta mudanças / tira do stage. |
| git mv ANTIGO NOVO | Move/renomeia arquivo ou pasta. |
| git rm ARQUIVO | Remove arquivo do repositório. |
| .gitignore | Define arquivos/pastas a ignorar. |
| git tag -a vX.Y -m "msg" | Cria tag anotada; use git push origin --tags para enviar. |
| git config --global alias.xxx | Cria atalhos (aliases) úteis. |
| git stash / pop / apply | Guarda e recupera alterações temporárias. |
| git push --force-with-lease | Push seguro após rebase (reescrita de histórico). |