Git – Sistema de versionamento de código. Registra o histórico de código.

Commit – registro de fotografia do diretório (snapshot)

\*Deve estar na main...

git commit

---

Ramos no Git – Ramifique cedo, ramifique sempre. Um ramo diz, em essência, “Quero incluir o trabalho deste commit e de todos seus ancestrais

Isso permite deixar a main em um commit e ramificar...

git branch <nome> - para criar uma ramificação no mesmo local

git checkout <nome > - para selecionar o ramo

git commit – para fazer um commit usando o ramo selecionado.

---

Branches e Merge

Como unificar trabalho de dois ramos diferentes? Esse conhecimento garante ramificar, desenvolver um novo recurso, e combiná-lo de volta.

Para combinar trabalhos, temos:

git merge <nome do ramo que irá commitar> – cria um commit especial que possui dois pais únicos. Em essência, significa: “Quero incluir todo o trabalho deste pai aqui com o daquele outro pai ali, e com o do conjunto de todos os seus ancestrais.”

git checkout -b <nome> - cria um ramo (Branch) com o nome de interesse e já seleciona o ramo para seguir com o commit

---

Rebase – Usado para construir uma sequência mais bonita e linear de commits. O registro de commits ficará mais limpo se for utilizado apenas rebase em vez de merge.

git checkout <nome do ramo>  
git rebase main – traz o ramo selecionado para frente do commit da main.

---

HEAD -> A cabeça aponta pro commit mais recente...

git checkout – para movimentar entre os commits... Aceita referenciação com ^ e ~[num] para subir a quantidade de vezes do referenciado...

git Branch -f main HEAD~3 – Move à força o ramo main 3 ancestrais acima do HEAD

---

git reset – reverte mudanças movendo para trás no tempo a referência do ramo. Funciona muito bem para ramos locais do próprio computador. Mas não para remotos. Para compartilhar mudanças com os outros, precisa-se usar o git revert.

Ex.: git reset HEAD~1 (em um cenário com 3 commits (0, 1 e 2), tem-se o retorno da main para C1.

---

Git revert HEAD – surge um novo commit (EXEMPLO C2’), que introduz mudanças – ele reverte exatamente aquelas do commit C2. Com o revert, você pode fazer push das suas mudanças para compartilhá-las com os outros.

---

Git cherry-pick <commit1> <commit2> <...> - copia uma série de commits abaixo do seu local atual (HEAD).

---

Rebase interativo do git – melhor forma de rever uma série de commits sobre os quais você está prestes a fazer um rebase...

Rebase -i – exibe uma interface para mostrar os commits prestes a serem copiados abaixo do alvo do rebase.

É possível reordenar os commits, ocultar commits (pick) e esmagar (squash – para combinar commits)

Git rebase -i HEAD~4 (onde há 6 commits (0,1,2,3,4,5 com main\* no 5).

Git rebase <nome> main – traz a main para a frente do commit <nome>

---

Git tag <nome\_tag> <commit> - fazem uma espécie de ancoragem e milestone para referenciar...

Git describe – mostra sua relação em relação a tag mais próxima...

Git describe <ref>, sendo <ref> qualquer coisa que sirva de referência. Sem explicitar, o git usa o HEAD... A saída tende a ser algo como: <tag>\_<numCommits>\_<g<hash>, sendo a tag ancestral mais próxima no histórico, o número de commits de distância da tag e o hash do commit sendo descrito.

Git checkout main^2 – faz especificar o segundo pai...

Git Branch <Nome> main^^2^ - HEAD sai da main 1x, vai para o segundo pai (^2), vai pra um commit mais ancestral, e cria um ramo com <nome> escolhido

---

Repositórios Remotos

Git clone – cria cópia local dos repositórios remotos.

Git Fetch – baixar dados de um repositório remoto. Baixa os commits que o repositório remoto possui que estejam faltando no repositório local e atualiza a referência para o qual os ramos remotos apontam. ELE NÃO MUDA NADA DO ESTADO LOCAL DO REPOSITÓRIO. NÃO ATUALIZA O RAMO MAIN, NEM MUDA NADA NA FORMA COMO O SISTEMA DE ARQUIVOS ESTÁ. É um passo de download...

Git Pull – Faz o fetch e o merge automaticamente. É um caminho mais curto para executar, por exemplo, git fetch; git merge o/main;

Git push – sobe as alterações do repositório local para o repositório remoto

Primeiro, vamos abordar o git push. Você aprendeu na lição sobre remote tracking que o Git descobria o repositório remoto *e* o ramo correspondente onde fazer o push olhando nas propriedades do ramo que está atualmente em checkout (verificando o ramo remoto que ele "segue"). Este é o comportamento quando nenhum parâmetro é especificado, mas o git push pode opcionalmente receber parâmetros na seguinte forma:

git push <repositório remoto> <lugar>

git push origin main – vá ao ramo main localmente, pegue todos os commits, vá ao repositório remoto chamado origin e coloque qualquer commit que estiver faltando e me diga quando estiver pronto. Quando coloca-se main, ele diz ao git onde os commits vão vir e para onde irão. Esse lugar seria onde sincronizar entre os 2 repositórios (local e remoto).

git push origin <origem>:<destino> - pode especificar tanto a origem como o destino...

Se usar git push origin com parâmetro vazio como

Git push origin :<nome> - vai deletar o ramo com nome

Se usar git pull origin :<nome> - ele cria um novo ramo com o nome...

git pull origin foo é igual a:

git fetch origin foo; git merge o/foo

git pull origin bar:bugFix é igual a:

git fetch origin bar:bugFix; git merge bugFix

Python: