**ANOTAÇÕES DOS COMANDOS USADOS NO GIT**

**Git status =** Exibe o estado do diretório de trabalho e o instantâneo preparado. Utilize essa opção em conjunto com os comandos git add e o git commit para ver com precisão o que vai ser incluído no próximo instantâneo.

**git add =** Move as alterações do diretório de trabalho para a área de staging.

**Git branch =** Esse comando é a ferramenta de administração de ramificações de uso geral. Permite criar ambientes de desenvolvimento isolados em um único repositório.

**Git checkout =** Além de verificar commits antigos e revisões de arquivos antigos, o git checkout também é a forma de navegar pelas ramificações existentes.

**Git clean =** Remove os arquivos não monitorados do diretório de trabalho. Esse é o correspondente lógico do git reset, que (em geral) só opera em arquivos monitorados.

**Git clone =** Cria cópia de um Repositório do Git existente.

**Git commit =** Faz um commit do instantâneo preparado no histórico do projeto.

**Git commit –amend =** Informação da marcação -- amend ao git commit permite corrigir o commit mais recente. Isso é muito útil quando você esquece de preparar um arquivo ou para omitir informações importantes da mensagem do commit.

**Git config =** Uma maneira conveniente de definir opções de configuração para a instalação do Git. Em geral, sua utilização só é necessária de modo imediato após a instalação do Git em máquina de desenvolvimento nova.

**Git fetch =** A busca faz o download a partir da ramificação de outro repositório, junto com todos os commits e arquivos associados. Mas, não tenta integrar nada em o repositório local. Assim, você tem a oportunidade de inspecionar as alterações antes de fazer o merge no projeto.

**Git init =** Inicializa novo repositório do Git. Se você quiser colocar um projeto sob controle de revisão.

**Git log =** Permite explorar as revisões anteriores de determinado projeto. Ele oferece várias opções de formatação para exibir o commit de instantâneos.

**Git merge =** Uma maneira poderosa de integrar alterações de ramificações divergentes. Após bifurcar o histórico do projeto com o git branch, o git merge permite unificar o histórico de novo.

**Git pull =** É a versão automatizada do git fetch. Faz o download de ramificação de um repositório remoto e faz a mesclagem imediata na ramificação atual.

**Git push =** É o oposto de buscar (com algumas ressalvas). Permite mover uma ramificação local para outro repositório, o que é um modo conveniente de publicar as contribuições. É semelhante ao svn commit, mas envia uma série de commits em vez de um único conjunto de alterações.

**Git rebase =** Permite mover ramificações, o que ajuda a evitar commits de merge desnecessários. A história linear resultante é, em geral, muito mais fácil de entender e explorar.

**Git rebase –i =** A marcação -i é usada para iniciar uma sessão de rebase interativa. Ela apresenta todos os benefícios de um rebase normal, porém, dá a oportunidade de adicionar, editar ou excluir commits durante o processo.

**Git reflog =** O Git mantém o controle das atualizações na ponta das ramificações usando um mecanismo chamado reflog. Ele permite retornar aos conjuntos de alterações mesmo que não estejam referenciados por nenhuma ramificação ou marcação.

**Git remote =** Uma ferramenta prática para administrar conexões remotas. Em vez de passar o URL completo para os comandos fetch, pull e push, ele permite usar um atalho mais significativo.

**Git reset =** Desfaz as alterações nos arquivos no diretório de trabalho. A redefinição permite limpar ou remover por completo as alterações que não foram enviadas a um repositório público.

**Git revert =** Desfaz o commit de um instantâneo. Ao identificar um commit com falha, uma maneira fácil e segura de realizar a sua remoção da base de código é por meio da reversão.

Fonte : https://www.atlassian.com/br/git/glossary