**GIT ADD - Adiciona o conteúdo do arquivo ao índice**

Esse comando atualiza o índice usando o conteúdo atual encontrado na área de trabalho para preparar o conteúdo testado para a próxima confirmação. Ele normalmente adiciona o conteúdo atual dos caminhos existentes como um todo, mas com algumas opções também pode ser usado para adicionar conteúdo com apenas parte das alterações feitas nos arquivos da área de trabalho aplicados ou remover caminhos que não existem na área de trabalho não mais.

Caso você queira começar a controlar o versionamento dos arquivos existentes (diferente de um diretório vazio), você provavelmente deve começar a monitorar esses arquivos e fazer um commit inicial. Você pode realizar isso com poucos comandos git add que especificam quais arquivos você quer monitorar, seguido de um commit:

$ git add \*.c  
$ git add README  
$ git commit -m 'initial project version'

**GIT COMMIT - Registra as mudanças no repositório**

Você tem um repositório Git e um checkout ou cópia funcional dos arquivos para esse projeto. Você precisa fazer algumas mudanças e fazer o commit das partes destas mudanças em seu repositório cada vez que o projeto atinge um estado no qual você queira gravar.

Armazena o conteúdo atual do índice em um novo commit junto com uma mensagem de log do usuário descrevendo as alterações.

O conteúdo a ser adicionado pode ser especificado de várias maneiras:

1- usando *git add* para incrementar "add" alterações no índice antes de usar o comando *commit* (Nota: até arquivos modificados devem ser "adicionados");

2-usando *git rm* para remover arquivos da área de trabalho e do índice, novamente antes de usar o comando *commit* ;

3- listando os arquivos como argumentos para o comando *commit* (sem --interactive ou --patch switch), caso em que a confirmação ignorará as mudanças testadas no índice e, em vez disso, registrará o conteúdo atual dos arquivos listados (que já devem ser conhecidos para Git);

4-usando a opção -a com o comando *commit* para "adicionar" alterações automaticamente de todos os arquivos conhecidos

5-usando os switches --interactive ou --patch com o comando *commit* para decidir um por um quais arquivos ou partes devem ser parte da confirmação além do conteúdo no índice, antes de finalizar a operação.

**GIT PUSH - Atualizar referências remotas junto com objetos associados**

Atualiza refs remotos usando refs locais, enquanto envia objetos necessários para completar as referências dadas.

Você pode fazer coisas interessantes acontecerem em um repositório a cada vez que você fizer isso, configurando *ganchos* lá.

<repositório> O repositório "remoto" que é o destino de uma operação de envio. Esse parâmetro pode ser um URL, ou o nome de um controle remoto

<refspec>… Especifique qual ref de destino para atualizar com qual objeto de origem.

O <src> é frequentemente o nome do ramo que você deseja enviar

**GIT PULL - Busca e integração com outro repositório ou um ramo local**

Incorpora as alterações de um repositório remoto no ramo atual. Em seu modo padrão, git pull é um atalho para git fetch seguido por git merge FETCH\_HEAD.

Mais precisamente, *git pull* executa *git fetch* com os parâmetros fornecidos e chama *git merge* para mesclar os cabeçalhos recuperados na ramificação atual. Com --rebase ele, ele executa o *rebase git em* vez de *git merge* .

**GIT STATUS - Mostra o status da área de trabalho**

Exibe caminhos que possuem diferenças entre o arquivo de índice e a consolidação HEAD atual, caminhos que possuem diferenças entre a área de trabalho e o arquivo de índice e caminhos na árvore de trabalho que não são rastreados pelo Git .O primeiro é o que você *iria* começar executando git commit; o segundo e o terceiro são o que você *pode* começar executando *git add* antes de executar git commit.

Saída desse comando foi projetada para ser usada como um comentário de modelo de confirmação. O formato longo padrão é projetado para ser legível, detalhado e descritivo. Seu conteúdo e formato estão sujeitos a alterações a qualquer momento.

Os caminhos mencionados na saída, ao contrário de muitos outros comandos do Git, são feitos em relação ao diretório atual se você estiver trabalhando em um subdiretório (isto é propositalmente, para ajudar a cortar e colar)

**Comando de prompt**

Prompt de comando é um recurso do Windows que fornece um ponto de entrada para digitar comandos do MS‑DOS (Sistema Operacional de Disco da Microsoft) e outros comandos do computador,ao digitar comandos, você pode executar tarefas no computador sem usar a interface gráfica do Windows.

**mkdir**

Cria uma pasta.

MKDIR [unidade:]caminho

Se as extensões de comando estiverem ativadas, MKDIR será alterado como a seguir:

MKDIR cria quaisquer pastas intermediárias no caminho, se necessário.

Por exemplo, se \a não existisse:

mkdir \a\b\c\d

seria o mesmo que:

mkdir \a

chdir \a

mkdir b

chdir b

mkdir c

chdir c

mkdir d

que seria o que você teria de digitar se as extensões estivessem desativadas.

**cd**

O comando CD significa "mudar diretório" e permite que abra diferentes diretórios ou pastas no disco rígido.

Clique em "Iniciar" e digite "CMD" na janela de busca na parte inferior do menu iniciar. Aperte "Enter" para abrir uma janela de prompt de comando

Digite "CD/" e aperte "Enter" para navegar até o diretório principal do drive C. "CD/" sempre te trará de volta ao diretório principal.

Digite "CD" e o nome do diretório que deseja abrir, e aperte "Enter" para navegar àquele diretório na janela do prompt de comando. Por exemplo, para abrir a pasta arquivos de programas, digitaria "CD Arquivos de Programas".

Navegue pelo diretório usando o comando "CD," junto com o nome de um subdiretório, para cavar mais fundo do sistema de arquivos. Se souber o caminho do arquivo exato para o qual deseja navegar, pode digitá-lo após o comando "CD." Por exemplo, para abrir o diretório de ajuda do Windows, digite "CD Windows/Ajuda" e aperte "Enter."

Digite "CD.." e aperte "Enter" para voltar ao diretório anterior.

**IF**

Executa o processamento condicional nos programas em lotes.

C:\WINDOWS\system32\cmd.exe

IF [NOT] ERRORLEVEL número comando

IF [NOT] seqüência\_de\_caracteres1==seqüência\_de\_caracteres2 comando

IF [NOT] EXIST nome\_de\_arquivo comando\_

**NOT**  Especifica que o Windows XP só deve executar o

comando se a condição for falsa.

**ERRORLEVEL número**  Especifica uma condição verdadeira se o

último programa executado retornar um código de

saída igual ou maior que o número especificado.

**seqüência\_de\_caracteres1==seqüência\_de\_caracteres2** Específica uma

condição verdadeira se as seqüências de texto

especificadas forem correspondentes.

**EXIST nome\_de\_arquivo** Especifica uma condição verdadeira se o nome de

arquivo especificado existir.

**comando**  Especifica o comando a ser executado se a condição

for atendida. O comando pode ser seguido pelo

comando ELSE, que executará o comando após a

palavra-chave ELSE se a condição especificada

for FALSA