**COMANDOS E DICAS GIT**

**Configura o nome do usuário no git.**

git config --lobal user.name “Waldson Neves”

**Configura o email do usuário no git.**

git config --global user.email “[waldsonfagundes@gmail.com](mailto:waldsonfagundes@gmail.com)”

**Listar todas a configurações do git.**

git --list

**Limpar tela**

clear

**Listar o que tem dentro da pasta.**

dir

* Repositório é onde o projeto está armazenado.

**Iniciar um repositório git vazio.**

git init

* Ramificação é o mesmo que brench e (master) é a ramificação principal do projeto.

**Forçar listagem de arquivos ocultos.**

dir -a

* Se digitar git init novamente o repositório reinicia.

**Sai da pasta.**

cd ..

* Podemos iniciar repositório com arquivos pré-existentes

**Mostra o estado dos arquivos que estão dentro da pasta de repositório, ex.: se estão marcados ou não para gravar alterações.**

git status

* ‘git status’ mostra a diferença do que está na área de preparação.

**Adiciona um arquivo ao git (rastrear as mudanças no arquivo).**

git add readme.txt

* Se der ‘tab’ 2x depois do ‘git add’ ele busca o arquivo dentro da pasta e preenche. (rastrear as mudanças no arquivo)

**Adiciona todos os arquivos de dentro da pasta.**

git add –all ou git add -A ou git add .

* Committed quase o mesmo que salvar.

**Salva uma nova versão no banco de dados.**

git commit -m “Aqui você colocar um título relevante”

**Mostra a diferença do repositório de trabalho e do último commit.**

git diff

* Quando o comando ‘git diff’ retornar o próprio pront de comando sem nenhuma resposta experimente ‘git reset’ isso vai desfazer as últimas alterações do índice do staging.

**Mostra a diferença na área de preparação.**

git diff --cached

**Mostra as configurações.**

git diff --staged

**Mostra a listagem de todos o commits que foram feitos.**

git log

* O head aponta para o último commit.

**Mostra todas as modificações em uma única linha.**

git log --oneline

* A execução do comando ‘git log’ pode ficar confusa quando existem muitos registros a serem

mostrados. Felizmente, esse comando possui algumas variações que podem facilitar a

visualização.

**Faz a listagem de logs limitado a N registros.**

git log -n

ex.: git log -2

**Mostra as mudanças de cada arquivo de acordo com o commit. É como se fosse feito um ‘git**

**diff’ em cada commit.**

git log -p

**Mostra a quantidade de alterações em cada arquivo em formato visual, sem mostrar as mudanças específicas.**

git log --stat

**Mostra somente a quantidade de alteração de forma menos visual que --stat.**

git log --shortstat

**Mostra a lista de arquivos modificados em cada commit.**

git log --name-only

**Mostra a lista de arquivos modificados em cada commit e para cada arquivo mostra as letras M, A**

**ou D.**

**M - Arquivo modificado**

**A - Arquivo adicionado**

**D - Arquivo removido**

git log --name-status

**Faz exatamente igual ao comando git log, porém mostra somente poucos caracteres do commit, ao invés de todos os 40.**

git log --abbrev-commit

**Mostra os commits com a data relativa comparado a data do dia atual. Ao invés de mostrar a data em que foi feita, mostra quanto tempo atrás foi feito.**

**Por exemplo,**

**"8 semanas atrás", "2 dias atrás"**

git log --relative-date

**Mostra a lista de commits juntamente com alguns caracteres no início da linha que mostram**

**quando uma branch surgiu e quando houve um merge.**

git log --graph

**Faz a listagem de commits mostrando o hash abreviado e a mensagem usada para o commit.**

**Além disso, como o nome já diz, cada commit é mostrado em somente uma linha.**

git log --oneline

**Também é possível informar o seu próprio formato através do parâmetro ‘git log --pretty=format:codigo’**

**Abaixo, algumas das configurações disponíveis para serem usadas.**

**Mostra o hash do commit.**

%H

**Mostra o hash do commit abreviado.**

%h

**Mostra o hash do commit anterior.**

%P

**Mostra o hash do commit do commit anterior abreviado.**

%p

**Mostra o nome do autor do commit.**

%an

**Mostra o e-mail do autor do commit.**

%ae

**Mostra a data relativa ao dia de hoje. Há quanto tempo a alteração foi feita.**

%ar

**Mostra o comentário realizado no commit.**

%s

**Abaixo, alguns exemplos das configurações acima sendo utilizadas.**

git log --pretty=format:"%h - %p: Feito por %an"

git log --pretty=format:"%h: %ar - (%ae) %an"

**O comando git log --oneline pode ser escrito da seguinte maneira.**

git log --pretty=format:"%h %s"

**Navegar pelos estados anteriores, adiciona o número do commit no final não precisa ser todos os números.**

git checkout numeroDaCommit

* Caso crie um commit em um estado anterior deve se incluir esse commit em uma nova branch se não ela ficará perdida. Ex.: git branch nomeDaNovaBranch númeroDoCommitCriado (git branch dev2 546d6s).

**Volta para brench master.**

git checkout master

* BRANCH – É uma ramificação do projeto que permite que ele siga por ‘n’ caminhos deixando que funcionalidades sejam implementadas sem impactar a estabilidades de funções já existentes.

**Desfaz uma alteração em apenar um arquivo, se excluir o arquivo ele restaura ele também, mas não funciona para arquivos adicionados que não estão rastreáveis.**

git checkout arquivo.txt

**Retorna ao estado inicial do último commit.**

git reset (vai mostrar todos os arquivos que serão restaurados)

git –hard (vai executar o reste)

ou

git reset –hard (faz tudo de uma só vez)

**Força a remoção de arquivos que foram adicionados mesmo não estando adicionado ao repositório git (untracked).**

git clean -f

**Cria arquivo da extensão .gitignore (não pode ter nome só dá para configura no cmd) vai ser utilizar para ignorar arquivos do projeto. No Linux e no Mac em vez de ‘ren’ ser ‘mv’. Para fazer isso primeiro deve criar um arquivo com o conteúdo para o gitignore com um nome simples ex.: a.txt, depois no cmd faça a alteração para a extensão gitignore sem colocar nome (‘ren a.txt .gitignore’).**

ren nomedoarquivo.txt .gitignore

**Dentro do arquivo .gitignore escreva os arquivos que queira ignorar do projeto ai quando chamar o git status ele nem aparece (dica pegar .gitignore de um projeto já criado para evitar o retrabalho.) no site da github tem arquivos de .gitignore já prontos para todos os tipos de projeto inclusive para android. Pesquisar Gitignore github já cai na página.**

\*.xlsx **ignora tudo**

bla/\*.xlsx **exclua tudo dentro da pasta bla**

Teste.xlsx **ignora somente um arquivo específico**

**Clona um repositório.**

git clone nomeDoRepositorio/ novoRepositorio

no curso usei git clone “teste com GIT”/ testegit (ele criou a pasta testegit além de clonar o repositório)

* Quando o nome do arquivo for grande ou tiver espaços utilize aspas “ ”.

**Começando com o GitHub**

**Clona o repositório do github.**

git clone URL\_do\_repositório

**Comando que envia mudanças para o servidor.**

git push

* Arquivos que estão ‘Untracked’ não sob.

**Baixa as atualizações do servidor.**

git pull

* Opção ‘Star’ coloca os projetos de outros como favoritos.
* Opção ‘Watch’ vai te notificar de todas as conversações do projeto.
* Opção ‘Fork’ essa opção faz um clone para o seu perfil, traz todas as alterações também.
* Opção ‘Pull Request’ enviar suas alteração que foi feita no projeto forkeado para a aprovação do master.
* Fazer um Merge ‘Pull Request’ significa aceitar as mudanças.
* Issues – serve para relatar erros no projeto.
* Labels – estão dentro das ‘Issues’ é uma forma de rotular de forma gráfica as ‘Issues’.
* Milestone - estão dentro das Issues é uma forma de dar informações para ‘Issues’ ex.: informar que será resolvida na versão 3.0 e pode colocar data para tratamento.
* Caso queira resolver uma ‘Issue’ para alguém no momento que está fazendo ‘Pull request’ no campo comentário pode se usar ‘**Close #escolheAissueAssociada’**. Ex.: Close #2.
* Use o site DILLINGER.io para editar o arquivo Readme que é muito parecido com html.
* Mergear – significa jogar a Branch (ramificação) para a master.

**Lista as branch existentes.**

git branch

**Criar nova branch.**

git branch nomeDaBranch

**Mudar de branch.**

git checkout nomeDaBranchExistente

**Cria uma branch e loga nela.**

git checkout -b nomeDaBranch

**Envia a branch para o servidor (boa prática: deixe o nome da ‘branch’ do servidor igual à da local).**

git push --set-upstream origin nomeDaBranch

ou

git push -u origin nomeDaBranch

**Atualiza o repositório local.**

git pull

**Deletar branch local (não pode está checkado na branch que vai deletar).**

git branch -d nomeDaBranch

**Deletar branch local de forma forçada (não pode está checkado na branch que vai deletar).**

git branch -D nomeDaBranch

**Deletar branch do servidor.**

git push --delete origin nomeDaBranch

* Quando a branch é apagada só localmente você ainda pode recuperá-la através do ‘git checkout nomeDaBranchDeletada’ esse comando é inicialmente usando para trocar de ‘branch’ mas neste caso ele recupera a branch.

**Renomear uma branch local estando logado nela.**

git branch -m nomeNovo

**Renomear uma branch local não estando logado nela**

git branch -m nomeAtual nomeNovo

**Atenção: Não é possível renomear branch no servidor, mas tem como fazer uma gambiarra, segue:**

1 – faz um ‘git pull’ na branch que quer renomear no servidor

2 – muda o nome da branch localmente ‘git branch -m novoNome’

3 – deleta a branch do servidor ‘git push --delete origin nomeDaBranchDoServidor’

4 - faz um ‘git push -u origin branchComOnomeNovo’

**Mesclar as alterações (atenção você sempre está trazendo mudanças para a branch que você está checkado).**

git merge nomeDaBranchQueEstaPuxandoAsAlterações

* O ‘merge’ faz o ‘commit’ automático se não houve conflito.
* Sempre faça ‘git pull’ depois que criar um branch e antes de fazer commit.

**Kdiff3 -> ferramenta para auxiliar a resolução de conflitos de forma visual.**

**Comando para abrir as ferramentas baseadas no git.**

git mergetool

**Mostra as configurações.**

git config --global --list

**Adiciona a ferramenta Kdiff3.**

git config --global --add merge.tool kdiff3

**Configurando o Kdiff3.**

git config --global --add mergetool.kdiff3.path “localOndeEstaInstaladoOKdiff3”

Ex.: git config --global --add “C:\ProgramData\Microsoft\Windows\Start Menu\Programs\KDiff3”

Para achar o caminho do path primeiro localize a pasta depois **abra a propriedade** do arquivo depois **copie o destino**, já tive problema com o path porque copiei o caminho da pasta, mas não especifiquei o arquivo e como a pasta tinha o mesmo nome do arquivo fiquei dando voltas até entender o problema.

**Ainda na configuração do Kdiff3.**

git config --global --add mergetool.kdiff3.trustExitCode false

**Abrindo a ferramento kdiff3.**

git mergetool

* Depois do comando anterior vai se abrir o programa kdiff3 dentro dele você terá **3 sessões** onde a primeira **A (base)** é a sessão que mostra o arquivo antes da modificação a **B (local)** seria o arquivo destino e a **C (remote)** é de onde está vindo as mudanças.

**Comando que faz o add e o commit ao mesmo tempo.**

git commit -a -m “nome do versionamento”

* A tag é um ponteiro que marca um commit especifico, a tag HEAD marca o último commit.

**Comando que cria uma tag.**

git tag -a nomeTag -m “mensagem da tag”

Ex.: git tag -a v1.0 -m “Versão 1 após alterações de traduação"

**Lista as tag existentes.**

git tag

**Enviar a Tag para o servidor.**

git push origin nomeDaTag

* Atenção: As tags no Github aparece na aba “release” .

**Comando para navegar pelas tags.**

git checkout nomeDaTag

**Remover tag localmente.**

git tag -d nomeDaTag

**Remover tag no servidor.**

git push --delete origin v1.0

* Para colocar tag em um commit antigo basta navegar até ele e criar as tag e não esqueça de enviar para o servidor.

**Forma de adicionar tags em commits anteriores sem precisar navegar**

git tag -a nomeDaTag codigoDaTag -m “Mensagem”

**Caso: Você está trabalhando numa branch e tem que mudar para atender a alguma solicitação em outra branch mas você escreveu alguns códigos porem não está pronto para commitar e não quer fazer checkout porque perdera as informações já criadas. Nesse caso use:**

git stash

* Esse código acima vai salvar o estado da sua branch e permitirá que você mude de branch sem precisar commitar ou fazer checkout, ele pega as mudanças que estão no repositório local e joga em uma área na memória do GIT.

**Lista as stash (mudanças que foram salvas momentaneamente).**

git stash list

**Caso queira criar um stash com mensagem.**

git stash save “mensagem”

**Comando que reaplica a última stash que você fez.**

git stash apply

**Comando que aplica a última stash e remove da lista (sempre a mais recente).**

git stash pop

**Remove um stash específico da lista.**

git stash drop nomeDaStash(nome da stash é padrão)

ex.: git stash drop stash@{1}

**DESFAZENDO COMMIT**

**Comando que desfaz commit (perde as alterações do commit).**

git reset --hard HEAD~2

O número é a quantidade de commits que você quer desfazer.

Caso precise adicionar uma pequena mudança em um commit já existente que você acabou de lançar, faça a alteração no arquivo depois o ‘add .’ (o commit tem que estar na área local) ai você utiliza o comando abaixo:

git commit --amend

* Não tem como desfazer um commit no servidor, teria que versionar uma alteração por cima.

comando traz atualizações do servidor remoto para o servidor local, mas ele não aplica as alterações (commits, branchs, tags)

git fetch

o comando git pull é a junção do comando git fatch com o git merge

se usar o comando git fetch as alterações feitas diretas no servidor remoto não mostrara o commit

se fizer um commit direto no servidor remoto e depois usar o comando ‘git fetch’ no servidor local as alterações desse commit não aparecerão no log usando o comando ‘git log --oneline’ mas aparecera se usar o comando git log origin/nomeDabranch --oneline isso significa que essa alteração esta em um estado intermediário ela já esta no repositório porém não esta aplicada.

para aplica as alterações do ‘git fetch’ deve se usar o ‘git merge’

uso do software: sourcetreeapp.com

conceitos de mensagens dentro do bash quando esta tentando mergiar

-Fast-forward

(quando consegue pegar os commits que estão em uma branch e aplicar diretamente na que esta fazendo merge) no exemplo do curso os commits que estavam na outra branch foram aplicadas na master não gerando conflito e deixando a linha gráfica sem ramificação porque o arquivo dentro das branchs são basicamente iguais.

-Auto-merging nomeDoArquivo.txt

-Merge made by the ‘recursive’ strategy

(essas mensages aparece quando a branch que esta fazendo o merge já sofreu commits durante o(os) commits da branch que esta sendo chamada. Para resolver esse problema o git gera um novo commit que junta os commits da duas branchs) então se a primeira branch que chamou o merge tinha e a que foi chamada tinha mais foi criado mais um ficando com 3 commits.

Rebase - será usado quando você quiser ter um histórico mais linear do projeto. Ele traz os commites de outras branch. Ex. loga-se na Branch1 e chama o ‘git rebase’ isso trará os commits da branch master e depois loga -se na branch master e faz o rebase chamando os commits da Branch1, usou o rebase duas vezes nesse caso um em cada branch igualando os commits.

Comando

git rebase nomeDaBranch

quando chamar o ‘git log --oneline’ vai aparecer os commits da branch mais os commites que foram trazidos pelo ‘git rebase’.

No fim o rebase e o merge fazem a mesma coisa, a diferença é como eles fazem isso o ‘merge’ faz a criação de um novo commit e junta os commits existentes o ‘rebase’ pega os commites da branch e aplica em cima da branchMaster esso deixa uma sequência linear no histórico de commites já o ‘merge’ vai mostrando as ramificações tornando difícil a leitura.

3 ESTADOS DO GIT

WORKING DIRECTOTY (diretório de trabalho) – todas as mudanças ficam aqui

- Modificados

- Excluidos

- Adicionados

Quando damos o git add jogamos eles para o próximo estado.

ALIAS - abreviações para fazer os comandos

Comando para adicionar atalho

git config --global alias.letraOuPalavra nomeDoComando

no curso testamos ‘git config --global alias.s status

Comando para remover o atalho

git config --global --unset alias.nomeDoAtalhoExistente

no curso testamos ‘git config --global --unset alias.s

Mostra qual é a url que está associada ao projeto

git remote -v

GREP -> adiciona um filtro para gerar uma lista

para listar branchs com grep

git branch | grep iniciaisDasbranchs

para listar tags com grep

git tag | grep iniciaisDasbranchs

Ferramenta visual para manipular Git

-GitKraken

Pull request -> serve para fazer uma revisão antes de mergia geralmente utilizada em alterações sensíveis no projeto.

DICA: Durante o processo de produção em equipe na criação do pull request o nomeie com “PR: nomeVersionamento” desse modo os PR: em meu nome eu não irei mergia e ficara para outra pessoa conferir.

STAGING AREA (staged) (área de preparação) – significa que o documento foi adicionado e preparados para serem versionados (commitados)

Quando damos o git commit jogamos vamos para o próximo estado.

COMMITTED (Salvar) – aqui salvamos os arquivos as alterações

NOTA: Bitbucket é outro gerenciador de repositorio remoto assim como o GitHub. O Bitbucket pertence a empresa Atlassian que fornecer outra aplicação chamada JIRA que serve para gerenciamento de tarefas.

Entendendo SCM - Source Code Management

É como os projetos são gerenciados em equipe, temos a branch master e em paralelo a branch develope e ramificando da branch develope temos as branch features, na branch master mantemos o código estável e com as tags de versoes

editor do bash

quando cair nele aperta aaa 3x para conseguir editar depois ‘esc’ e depoi ‘ : ’ para escrever ‘wq’ significa write quit

wq -> sai de umas partes onde o git bash exige a mensagem, você responde primeiro da ‘esc’ e depois ‘wq’ para sair