**USANDO O GIT**

**NO SITE DO GITHUB**

**Branch** -> (galhos) usados para fazer cópias do projeto e desenvolver funcionalidades, podemos criar quantos branch quisermos

**Commits** -> alterações realizadas não anexadas ao index com os commits (confirmação de alterações)

**Pull request** -> propor mudanças para o branch principal, nesta faze podemos revisar, analisar e propor discussões antes de realizar o merge. Podemos colocar feedbacks nos pull request.

**Merge** -> mesclar/atualizar os arquivos entre 2 branchs, podemos deletar o branch que não for principal depois.

## FLUXO DE TRABALHO

Seus repositórios locais consistem em três "árvores" mantidas pelo git. A primeira delas é sua **Working Directory** que contém os arquivos vigentes. A segunda **Index** que funciona como uma área temporária e finalmente a **HEAD** que aponta para o último *commit* (confirmação) que você fez.

**INSTALAÇÃO**

* apt-get install git

Teve um erro quando fui instalar no debian, pesquisar

**CRIANDO UM NOVO REPOSITÓRIO**

Crie uma nova pasta, abra-a e execute o comando abaixo para criar um novo repositório

* git init

.

**CONFIGURANDO O GIT**

git config --global user.email “seuEmail”

git config --global user.name “seuNome”

git config --list (Ver o que está configurado no git, **ele separa para cada usuário**)

## OBTER/ENVIAR ALTERAÇÕES PARA UM REPOSITÓRIO

Para clonar um repositório

* git clone /caminho/para/o/repositório

Quando usamos um servidor remoto, podemos acessar direto com o comando abaixo ou criar um “remote” para facilitar o acesso:

* git clone https://github.com/usuario/repositorio

Com as alterações no **HEAD** (depois de realizar algum commit) da sua cópia de trabalho local. Para enviar estas alterações ao seu repositório remoto, execute

* git push origin máster ou git push –u origin master

Altere ma*ster* para qualquer ramo (*branch*) desejado, enviando suas alterações para ele.

**Origin** é um remote criado automaticamente quando associamos ao repositório o nome pode ser mudado e também podemos criar vários remotes para diferentes sites Ex:

git remote add github [https://user](https://user/)@github.com/user/MeuProjecto.git

git remote add bitbucket [https://user](https://user/)@github.com/user/MeuProjecto.git

git remote add empresa [https://user](https://user/)@github.com/user/MeuProjecto.git

e para fazer um push em determinado repositório másters

* git push –u bitbucket máster (**ou branch que quiséssemos**)

Se você não clonou um repositório existente e quer conectar seu repositório a um servidor remoto, você deve adicioná-lo com

* git remote add origin <servidor>

Para alterar a URL

* git remote set-url origin <servidor>

Para listar os remotes

* git remote

Para deletar um remote

* git remote remove <nomeRemote>

Renomear remote

* git remote rename <oldName> <newName>

## ADICIONAR ARQUIVO E COMMITAR

Você pode propor mudanças (adicioná-las ao **Index**). Depois de realizar alterações nos arquivos do projeto devemos propor as alterações usando:

* git add <arquivo>
* git add \* ou git add –all

fazer inclusões interativas

* git add -i

**Obs:** se fizer alterções em algum arquivo depois de rodar o commando “git add” o arquivo não entrará o commit para inclui-lo no commit deve-se rodar o comando “git add arquivo” para incluílo no commit

**Obs2:** podemos pular a área de seleção “git add” passando o parâmetro “a” para o commit, com isso o commit seleciona todos os arquivos alterados antes de realizar o commit

* git commit –a –m “pulando a area de seleção (git add)”

**COMMIT**

Depois de adicionar os arquivos devemos revemos

* git commit -a (testar este)
* git commit –m “comentários das alterações”
* git commit –v (passa a saída do áster git diff para o commit)
* git commit (sem parâmetros passa a saída do comando git status)

Agora o arquivo é enviado para o HEAD, mas ainda não para o repositório remoto. Se desejarmos enviar para o remositório remoto devemos usar o

* git push <remote> <branch> (conforme descrito acima)

**REMOVENDO ARQUIVOS**

* git rm -f arquivo (deleta arquivo)
* git rm –cached arquivo (tira arquivo do index mas, mantem arquivo)

**MOVER RENOMEAR ARQUIVOS NO GIT**

* git mv arquivo.origem arquivo.destino

## RAMIFICANDO

*Branches* (“ramos”) são utilizados para desenvolver funcionalidades isoladas umas das outras. O branch *máster* é o branch “padrão” quando você cria um repositório. Use outros branches para desenvolver e mescle-os (*merge*) ao branch máster após a conclusão.

**Git branch -h → lista ajuda simples**

**git branch --help → man git branch**

Ver qual branch

* git branch

Para criar

* git branch nomeBranch

Mudar de branch

* git checkout nomeBranch

Cria um branch e muda para ele

* git checkout –b nomeBranch

Para remover um branch

* git branch –d nomeBranch

um branch *não está disponível a outros* a menos que você envie o branch para seu repositório remoto  
git push origin <nomeBranch>

## ATUALIZAR & MESCLAR

Para atualizar seu repositório local com a mais nova versão, execute

* git pull
* git pull remote <branch>

na sua pasta de trabalho para *obter* e *fazer merge* (mesclar) alterações remotas.  
para fazer merge de um outro branch ao seu branch ativo (ex. máster), Selecione o branch que deseja e use o comando abaixo:

* git merge <branch>

em ambos os casos o git tenta fazer o merge das alterações automaticamente. Infelizmente, isto nem sempre é possível e resulta em *conflitos*. Você é responsável por fazer o merge estes *conflitos* manualmente editando os arquivos exibidos pelo git. Depois de alterar, você precisa rodar o comando abaixo para mostrar que os conflitos foram resolvidos

* git add <arquivo>

Para ver quais arquivos estão com conflito rode:

* git status

O git colocar um marcador dentro dos arquivos em conflito para mostrar onde está a diferença entre os arquivo para podermos decidir qual modificação deve ficar

antes de fazer o merge das alterações, você pode também pré-visualizá-as usando

* git diff <branch origem> <branch destino>

Temos ainda a ferramenta visual para resolver os conflitos:

* git mergetool

## ROTULANDO

É recomendado criar rótulos para releases de software. Este é um conhecido conceito, que também existe no SVN. Você pode criar um novo rótulo chamado *1.0.0* executando o comando

* git tag 1.0.0 1b2e1d63ff

o *1b2e1d63ff* representa os 10 primeiros caracteres do id de commit que você quer referenciar com seu rótulo. Você pode obter o id de commit com

* git log

você pode também usar menos caracteres do id de commit, ele somente precisa ser único.

## SOBRESCREVER ALTERAÇÕES LOCAIS

No caso de você ter feito algo errado (que seguramente nunca acontece ;) você pode sobrescrever as alterações locais usando o comando

* git checkout -- <arquivo> (para descartar alterações depois de adicionar o arquino no index)

isto substitui as alterações na sua árvore de trabalho com o conteúdo mais recente no HEAD. Alterações já adicionadas ao index, bem como novos arquivos serão mantidos.

Se ao invés disso você deseja remover todas as alterações e commits locais, recupere o histórico mais recente do servidor e aponte para seu branch máster local desta forma

* git fetch <nomeBranch>
* git reset –hard origin/master
* git reset HEAD <file> → to unstage

**PROXY NO GIT**

* git config –global http.proxy meu.servidor:8080
* git config –global http.proxy utilizador:password@meu.servidor:8080
* git config –global http.sslverify false (se der erro no proxy)

para desconfigurar

* git config --global –unset http.proxy

**Verificando status de seus arquivos**

* git status

Para ver em detalhes as alterações e responder O que você alterou, mas ainda não selecionou (stage)? E o que você selecionou, que está para ser commitado? Utilizamos o comando abaixo

* git diff

visualizar históricos de commits

* git log

visualizar somente 2 saidas de log

* git log -2

Configura o “git log” para exibir os logs em apenas uma linha por commit

* git config format.pretty oneline
* git config –unset format.pretty (para desconfigurar)

visualizar somente 1 linha por commit

* git log --pretty=oneline

formatar sua própria saida de log

* git log --pretty=format:"%h - %an, %ar : %s"

| Opção | Descrição da saída |
| --- | --- |
| %H | Hash do commit |
| %h | Hash do commit abreviado |
| %T | Árvore hash |
| %t | Árvore hash abreviada |
| %P | Hashes pais |
| %p | Hashes pais abreviados |
| %an | Nome do autor |
| %ae | Email do autor |
| %ad | Data do autor (formato respeita a opção -date=) |
| %ar | Data do autor, relativa |
| %cn | Nome do committer |
| %ce | Email do committer |
| %cd | Data do committer |
| %cr | Data do committer, relativa |
| %s | Assunto |

* usar saídas do git coloridas  
  git config color.ui true

**Arquivo .gitignore**

podemos configurar o arquivo “.gitignore” para ignorar arquivos que não queremos que sejam monitorados. Ex de config

# um comentário – isto é ignorado

# sem arquivos terminados em .a

\*.a

# mas rastreie lib.a, mesmo que você tenha ignorado arquivos terminados em .a acima

!lib.a

# apenas ignore o arquivo TODO na raiz, não o subdiretório TODO

/TODO

# ignore todos os arquivos no diretório build/

build/

# ignore doc/notes.txt mas, não ignore doc/server/arch.txt

doc/\*.txt

**Inteface gráfica padrão para visualizar históricos**

* gitk