Github -> local para armazenamento de projetos versionados pelo git.

Git-> sistema para controle de versão

-------------------------------------------------------------------------------------------------------------------

INSTALANDO O GIT

\* No linux já vem instalado

-------------------------------------------------------------------------------------------------------------------

CONFIGURAÇÃO INICIAL DO GIT

O fit guarda as informações em 3 lugares (gitconfig do sistema, do usuario e do projeto)

\* Config do usuário conhecido como **global**

**$git config --global user.name “Aline Lopes”**

**$git config --global user.email “aline.marina94@gmail.com”**

*Não mostra nada, configura ocultamente*

CONFIGURAR EDITOR PRINCIPAL DO GIT

**$git config --global core.editor code .**

*DESCOBRIR INFORMAÇÕES DA VARIAVEL*

***$git config user.email*** *ou* ***user.name***

Saber todas as informações do git

**$git config --list**

-------------------------------------------------------------------------------------------------------------------

INICIALIZAR O PROJETO

Criar pastas

**$mkdir git-course**

Entrar na pasta

**$cd git-course**

Responsavel por inicializar o repositório e enxergar as modifcações do projeto

**$git init**

Entrar no .git criado

**$cd .git**

Pastas do .git

HEAD=branche padrão que ele está

Branches=quais os branches

Config= configuração do repositorio

Description=descrição

Hooks=gatilhos para certas ações

Info=

Objects=

Refs=

-------------------------------------------------------------------------------------------------------------------

O CICLO DE VIDA DOS STATUS DE SEUS ARQUIVOS

Separa em quatro estados bem definidos como os arquivos serão divididos

UNTRACKED= o arquivo acabou de ser adicionado no repositório, mas o git ainda não sabe que ele existe

UMMODIFIED= não foi modificado. Existe mas não houve modificaão.

MODIFIED= arquivo modificado

STAGED= momento em que o arquivo é armazenado para as versões.

Uma vez que ele foiadicionado no staged, o status altera para UMMODIFIED

**$git status**

**$git add git\_e\_github.docx**

Se alterar o arquivo antes do commit e quiser que as modificações sejam comitadas e necessário add novamente

**$git commit -m "Add tutorial até commit"**

Os números entre as chaves siginificam o id da versão.

Se comitar sem add dá erro, ele não é comitado.

-------------------------------------------------------------------------------------------------------------------

VISUALIZANDO LOGS

Mostrar informações sobre o commit

**$git log**

Informações sobre de que branche p que branche, quais tags foram adicionadas

**$git log --decorate**

Filtrar pelo autor

**$git log --author="Aline"**

Em ordem alfabetica mostra quem foram os autores, quantos commites fizeram e quais eles foram

**$git shortlog**

Quantidade de commites e a pessoa

**$git shortlog -sn**

Mostra em forma gráfica o que está acontecendo com o branches e versões

**$git log --graph**

Pela hash dá p saber o que foi modificado, o que foi commitado ou não

**$git show *0d7a9b8bc2493759d187c64335b996dfe31972d6***

*aline@aline-PC:~/git-course$ git log --graph*

*\* commit 0d7a9b8bc2493759d187c64335b996dfe31972d6 (HEAD -> master)*

*Author: “Aline <“aline.marina94@gmail.com”>*

*Date: Thu Aug 2 18:03:50 2018 -0300*

-------------------------------------------------------------------------------------------------------------------

VISUALIZANDO O DIFF

Visualizar as alterações do arquivo antes de dar commit.

**$git diff**

Saber qual os arquivos que foram modificados

**$git diff --name-only**

Quando se da commit em um arquivo que já existiu pode-se utilizar o *-am*(adicionar todos os arquivos modificados mais a mensagem) Não é preciso utilizar o *add.*

**$git commit -am "Edit arquivo"**

-------------------------------------------------------------------------------------------------------------------

DESFAZENDO COISAS

Resetar arquivo antes de add

**$git checkout git\_e\_github.docx**

Resetar arquivo depois do add. Tirar o arquivo do unstaged.

**$git reset HEAD git\_e\_github.docx**

*Unstaged changes after reset:*

*M git\_e\_github.docx*

Existem três tipos de reset que são:

--soft = pegar as moficações e ignorar o commit e vai deixar ele com o status unstaged, pronto para ser commitado novamente

--mixed = mata o commit e voltar os arquivo para o status mofied

--hard = ignorar a existência do commit e tudo que foi ,feito do commit.

Para fazer o reset, é necessário escolher sempre a hash do commit antes do deseja-se ignorar.

Exemplo:

Commit 3

Commit 2

Commit 1

Se quer ignorar o Commit 3, escolhe a hash do Commit 2.

**$git reset --soft 64573gt59** (64573gt59 representa o hash)

Obs: --hard não deve ser usado em repositórios compartilhados.

-------------------------------------------------------------------------------------------------------------------

CRIANDO O REPOSITÓRIO REMOTO

Ir no site do github e criar um novo repositório apertando o botão

*Não selecionar a opção para gerar um readme.mp pois ao se ter histórico local, o historico gerado por essa info será diferente. Mlehor evitar tal situação.*

-------------------------------------------------------------------------------------------------------------------

CRIANDO E ADICIONANDO UMA CHAVE SSH

Obs: Para problemas de push para o repositório remoto.

O SSH serve para autenticar o usuário no servidor. A chave privada presente na nossa máquina é utilizada para descriptografar a chave pública do servidor e assim permitir submeter o arquivo ao repósitorio remoto.

*GERAR A CHAVE SSH*

*Abrir o terminal e colocar*

ssh-keygen -t rsa -b 4096 -C "*your\_email@example.com*"

***Obs:*** *o email precisa ser exatamente igual ao do usuário*

Aparece o texto

Generating public/private rsa key pair.

Local para adicionar a chave. Geralmente deixa como padrão.

Enter a file in which to save the key (/home/*you*/.ssh/id\_rsa): *[Press enter]*

Quando pede um passaphrase e clica em enter. Não se preocupe com isso.

Enter passphrase (empty for no passphrase): *[Type a passphrase]*Enter same passphrase again: *[Type passphrase again]*

É possível encontrar as chaves SSH no diretório: cd ~/ .ssh/

Liste as chaves criadas com ls. A chave que iremos precisa possui o final .pub. Para utilizá-las é necessário:

**$cat id\_rsa.pub**

Assim terá acesso a chave. Copie a chave. Vá ao github.

Selecione no menu *settings.* Selecione SSH and GPG Keys. Aperte o botão SSH Key. Adicione a chave copiada e adicione um título.

-------------------------------------------------------------------------------------------------------------------

LIGANDO UM REPOSITÓRIO LOCAL A UM REMOTO

Informações para a ligação são dadas na pagina do repositório criado. Siga-as:

**$git remote add origin https://github.com/AlineLopes/git\_course.git**

**$git remote**

**$git remote -v**

O git push envia todas as informações para o repositório criado remotamente

**$git push -u origin master**

Ele diz que vai enviar os dados do respoitorio local(master) para o repositorio remoto(origin)

-------------------------------------------------------------------------------------------------------------------

ENVIANDO MUDANÇAS PARA UM REPOSITÓRIO REMOTO

**$git status**

**$git commit -am "Add instrucoes"**

**$git push origin master**

-------------------------------------------------------------------------------------------------------------------

CLONANDO REPOSITÓRIOS REMOTOS

O git clone permite você clonar todo o repositório para a sua maquina local. Ao clonar utilize o SSH, pois é mais rápido.

**$g clone [git@github.com:AlineLopes/git\_course.git](mailto:git@github.com:AlineLopes/git_course.git)**

Você consegue submeter as informações do repositório somente se ele for seu, caso contrario não poderá fazer isso.

-------------------------------------------------------------------------------------------------------------------

FAZENDO FORK DE UM PROJETO

O fork pega um projeto que não é seu e faz uma cópia dele para você. Tem um repositório que quero contribuir(algum erro, ou falta de info).

Primeiro faço um FORK, faço as mudanças e depois faço um pull request.

Aperte o botão FORK e escolhe o seu repositório.

-------------------------------------------------------------------------------------------------------------------

O QUE É UM BRANCH E POR QUE USAR?

Branch é um ponteiro móvel que leva um commit. Pode-se mais de um branch.

Porque usar?

Pode modificar sem alterar o local principal(master) - Posso consertar um bug sem ser no branch master, melchendo em outro lugar.

Facilmente “desligavel” - Pode-se criar e apagar branchs facilmente.

Multiplas pessoas trabalhando em diferentes branchs sem ninguem atrapalhar ninguem

Evitar conflitos, pois trabalha-se separadamente. No final se mesclar o branch separado com o branch master.

-------------------------------------------------------------------------------------------------------------------

CRIANDO UM BRANCH

**$git checkout -b testing**

Mostra os branchs existentes

**$git branch**

-------------------------------------------------------------------------------------------------------------------

MOVENDO E DELETANDO BRANCHs

Ir para o branch desejado:

**$git checkout testing**

Deletar um branch

**$git branch -D testing**

-------------------------------------------------------------------------------------------------------------------

ENTENDENDO O MERGE

Após a modificação do branch externo é necessário unir os branches. Existem duas formas bem diferentes de unir

Utilize sempre que puder o REBASE, quando se utiliza mtos branches

MERGE

Do inicio.

Criamos um novo commit na nova branch.

Para juntar os branches, cria-se um novo commit

Vantagem: operação não destrutiva, não movimenta o histórico, pois cria um novo commit, mas é um commit extra. Quando se faz um merge deixa-se o histórico poluído, devido a quantidade de branches.

PRATICA

Cria-se uma pasta

**$mkdir rebase-merge**

Entra na pasta

**$cd rebase-merge/**

Inicializa

**$git init**

Cria-se um arquivo e realiza um commit. Commit para o branch master.  
Cria-se um novo branch.

**$git checkout -b test**

**$vim bar**

**$git add bar**

**$git commit -m "add bar"**

Muda para o branch master

**$git checkout master**

**$vim buzz**

**$git add buzz**

**$git commit - m “add buzz”**

**$git log**

Ao observar o log, nota-se que o commit do test não está adicionado. Para fazer o merge desses branches, faz o seguinte:

**$git merge test**

**$git log --graph**

REBASE

Cada branch tem um commit separado para cada um. O rebase irá pega o branch externo e vai mover ele para frente do master, e assim o branchs apontaram para o mesmo commit nesse momento.

Vantagem: evita commite extra, histórico linear, pois trabalha com uma linha reta. Contudo, perde a ordem cronológica, pois ao adicionar o commit para o inicio da fila, mudando assim o histórico. Pode dar conflitos. Sugere-se usar $git pull --rebase, evitando conflitos.

**$vim fizz**

**$git add .**

**$git commit -m “add fizz”**

**$git log**

**$git checkout -b rebase-branch**

-------------------------------------------------------------------------------------------------------------------

GITGNORE

É um arquivo no diretório principal onde escrevemos o que ele não utilize. Especifica-se o arquivo e as extensoes.

**$ vi .gitgnore**

Ignorar extensões como um todo escrever no arquivo:

**\*.json**

Ignorar arquivos determinados

**db.xls**

Padrões para excluir certos tipos de arquivo

Github.com/github/gitignore/blob/master/

-------------------------------------------------------------------------------------------------------------------

GITSTACH

Responsável por guardar modificações não comitadas em um arquivo que possa chamar quando achar necessário.

Por exemplo preciso criar um novo branch, mas a modificação no meu projeto ainda não é a desejada para subeter ao commit.

**$git stash**

Ele vai guardar essa modificação em um arquivo WIP.

É aplicada as moficações guardadas no projeto.

**$git stash apply**

Listar todas as stash

**$git stash list**

Limpa o stash

**$git stach clear**

-------------------------------------------------------------------------------------------------------------------

CRIAR ATALHOS DE COMANDOS

**$git config --global alias.s status**

**$git config --global alias.l log**

-------------------------------------------------------------------------------------------------------------------

VERSIONANDO TAGS

Quando se trabalha em projetos muito grandes é importante organizar as versões. Exemplo agrupar commits referentes a criação de uma pagina.Para isso se cria tag.

**$git tag -a 1.0.0 -m “Login finalizado”**

Sobe as tags para o repositório

**$git push origin master --tags**

Elas são chamadas de releases. Podendo baixar o estado atual naquele release.

Cria-se a tag após o commit final de um arquivo ou projeto.

-------------------------------------------------------------------------------------------------------------------

GIT REVERT

Aplica-se quando se quer desfazer uma atualização.

**$git revert 8789dhgfidg** (hash do commit)  
  
Ele apaga a alteração, mas não apaga o commit.

Finalidade: Reverte as modificações e não perde do histórico as mudanças realizadas.

-------------------------------------------------------------------------------------------------------------------

APAGAR TAGS E BRANCHES NO REPOSITORIO REMOTO

APAGAR TAG LOCAL

**$g tag -d 1.0.1**

APAGAR TAG REMOTO

**$git push origin 1.0.0**

APAGAR BRANCH REMOTO

**$git push origin :test**

-------------------------------------------------------------------------------------------------------------------

RESOLVENDO CONFLITOS

-------------------------------------------------------------------------------------------------------------------

CRIANDO O SERVIDOR

No servidor: supondo que o projeto se chame “meuapp”

**$ mkdir meuapp**

**$ cd meuapp**

**$ git init --bare** // o --bare serve para inicializar um repositório vazio que será compartilhado com mais pessoas

Esses comandos inicializam o repositorio vazio para o projeto

Na máquina do programador: como o repositório no servidor está vazio, precisamos criar um repositório inicial e submeter este repositório ao servidor.

**$ mkdir meuapp**

**$ cd meuapp**

**$ git init**

Criamos alguns arquivos...

**$ git add .**

**$ git commit -m ‘arquivos iniciais do projeto’**

**$ git remote add origin programador@vps:/home/programador/git/meuapp**

**$git push origin master**

Estes comandos inicializam o repositorio local do projeto. Em seguida é adicionado o repositório remoto e finalmente feito o push. Depois disso basta que os outros usuários façam o clone do projeto.

Clonar repositório

**$git clone programador@vps:/home/programador/git/meuapp(caminho\_do\_repositório)**

Pull em repositório.

**$git pull origin master**

-------------------------------------------------------------------------------------------------------------------

MERGE DE MUDANÇAS ANTES DO PUSH

Depois do commit e a mudança no branch o outro colaborador avisa que submeteu alterações, logo é preciso fazer um merge para depois fazer um push

Baixar alterações

**$ git fetch origin**

Fazer merge em da sua mudança com as alterações utilizando o git merge

**$ git merge origin/featureBee**

Push do branch mesclado com branch do servidor

**$ git push origin featureB:featureBee**

Fazer o merge do trabalho do colaboraddor com o seu próprio branch

**$ git checkout featureA**

**$ git merge origin/featureA**

Fazer commit com outros branchs

**$git push origin featureA**