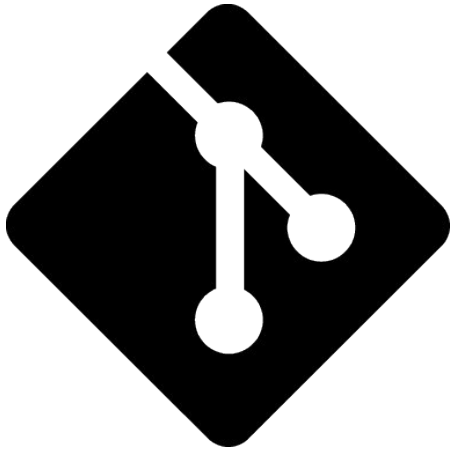


# Minicurso de Git e Github.

Ministrante : Pedro Ulisses Maia.

02 e 03 de Maio, 2023


# Planejamento do Minicurso na parte sobre Git.

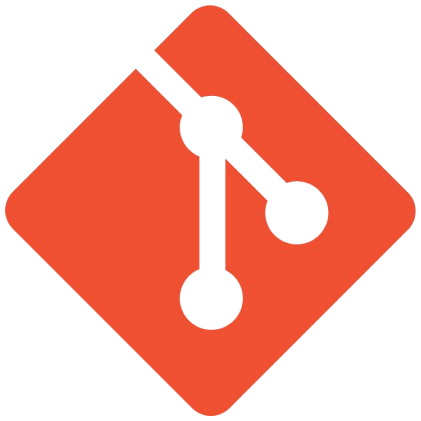
- Explicar sobre o que é um sistema versionador de arquivos;
- Como funciona um sistema versionador de arquivos;
- Um pouco da história dos sistemas versionadores ao logo do tempo;
- O que é Git , como surgiu o Git;
- Como o Git pode estar ajudando no desenvolvimento de um código/projeto , vantagens de usar;
- Como instalar na sua máquina;
- Configuração inicial do Git ;
- Termos que são usados ;
- Lista de linguagens mais usadas e projetos que usam Git para gerenciar as suas versões;
- Começando a parte prática de como mexer no Git , ou seja , vendo os comandos principais ou mais úteis do Git;
- Parte prática , fazer exercícios de fixação relacionado ao que foi abordado.

# Planejamento do Minicurso na parte de Github.

- O que é o Github;
- O que são repositórios remotos;
- O que é e como funciona um repositório remoto;
- Outros repositórios remotos além do Github;
- Um pouco sobre a história do Github e seu desenvolvimento;
- Projetos importantes que estão disponíveis no Github;
- Configurando o Git com Github através de uma chave SSH;
- Um pouco sobre as opções disponíveis no Github;
- Criando repositórios e fazendo prática dos comandos de Git com Github com exercícios de fixação;
- Finalizando.

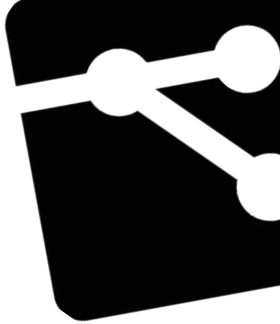
https://github.com/pedroulissespu/Slide-Minicurso

# Git



git

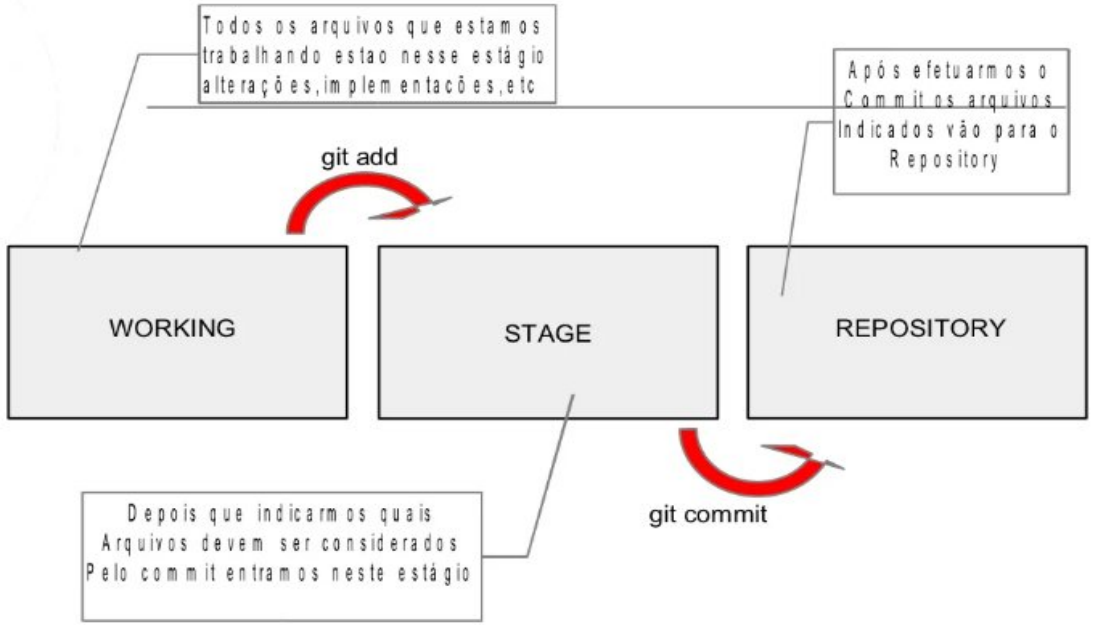
# 1. O que é um sistema versionador de arquivos ?

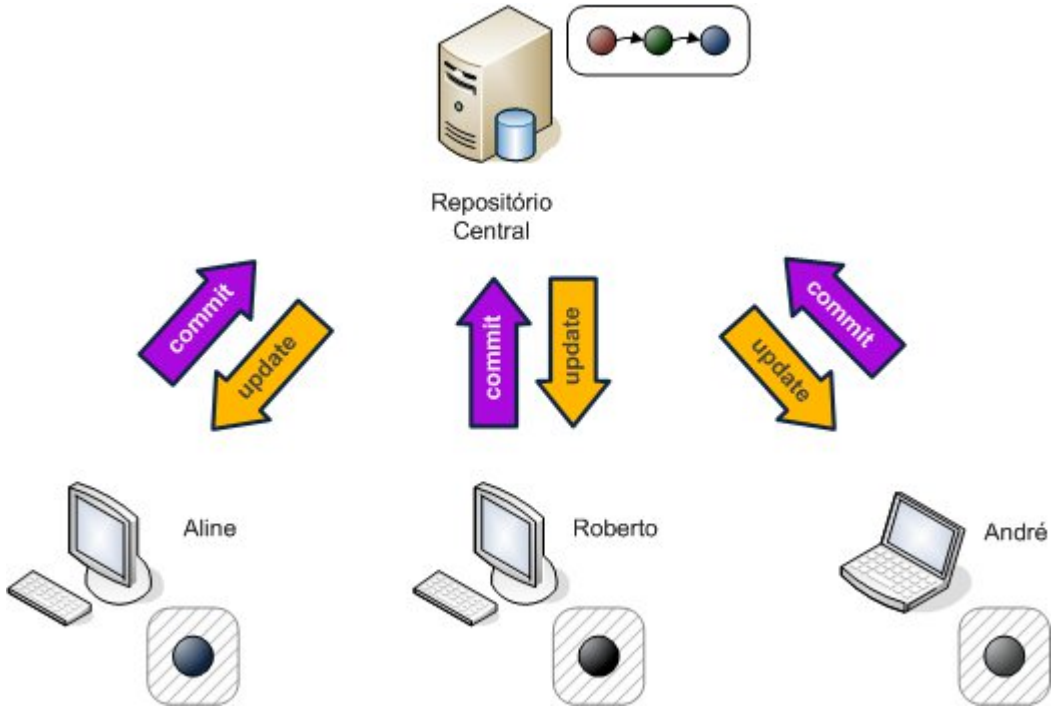


- Uma maneira de gerenciar arquivos e diretórios;
- Acompanhar as mudanças ao longo do tempo;
- Relembrar versões anteriores;
- Trabalhar em um projeto em conjunto.

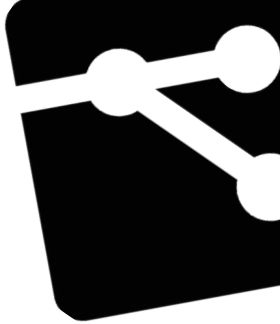
# 2. Como funciona um sistema versionador de arquivos.

- De modo geral , as imagens a seguir explica basicamente o funcionamento do Git , lembrando , somente o Git , não está sendo ainda citado o Github.



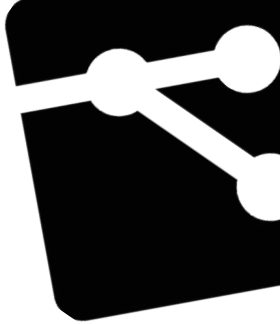



# 3. Um pouco da história dos sistemas versionadores de arquivos.



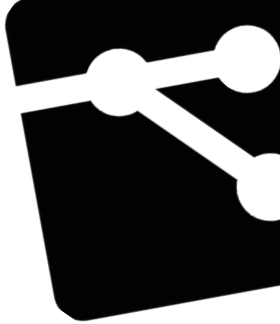
- (1972) Source Code Control System (SCCS) – Código Fechado;
- (1982) Revision Control System (RCS) – Código Aberto;
- (1985) Concurrent Versions System (CVS) – Código Aberto;
- (2000) Apache Subversion (SVN) – Código Aberto;
- (2000) BitKeeper SCM – Código Fechado.

# 3.1. SCCS – Source Code Control System ( 1972 )



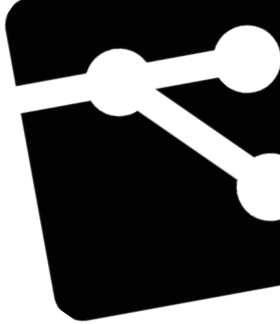
- Um sistema versionador de arquivo centralizado;
- Código Fechado/Proprietário;
- Escrito na linguagem C;
- Lançado pela Nokia Bell Labs ( Originalmente AT&T Bell Labs );
- Foi um dos primeiros sistema versionador de arquivos ser lançado e ficou sendo utilizado durante anos , porém tinha muitos problemas de consistência , velocidade e muito limitado.

# 3.2. RCS – Revision Control System. ( 1982 )



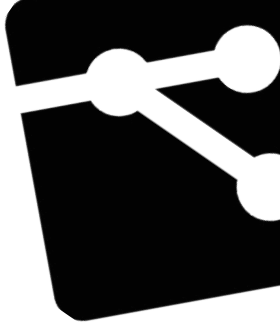
- Um sistema versionador de arquivo centralizado;
- Open Source;
- Escrito na linguagem C;
- Lançado por Walter F. Tichy para ser uma alternativa do SCCS;
- Não foi tão popular igual o SCCS e foi substituído pelo CVS anos depois , sua principal desvantagem era que não dava pra trabalhar com o projeto inteiro , somente com arquivos individuais , além é claro , da sintaxe complicada sendo de difícil o aprendizado.

# 3.3. CVS - Concurrent Versions System ( 1985 )



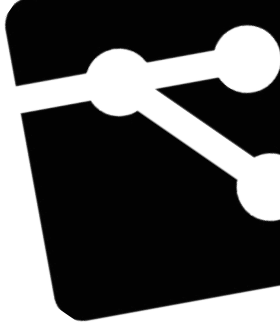
- Um sistema versionador de arquivo centralizado;
- Open Source;
- Escrito na linguagem C;
- Mais popular na época e ficou assim durante anos;
- Porém tinha problemas de consistência e velocidade.

# 3.4. SVN - Apache Subversion ( 2000 )



- Um sistema versionador de arquivo centralizado;
- Open Source;
- Escrito na linguagem C;
- Era muito parecido com o CVS , o motivo disso era que ele queria resolver/consertar problemas que o CVS possuía;
- Do mesmo jeito que foi um ponto positivo praticamente copiar o CVS , ao mesmo tempo era um problema pois a comunidade na época reclamava muito por conta disso e foi falado durante anos sobre;
- Ativo até os dias atuais( Última versão LTS : 1.14.2 , versão foi checada no dia 26/09/2022 );

# 3.5. BitKeeper ( 2000 )



- Distribuído;
- Proprietário;
- Escrito na linguagem C;
- Apesar de ser proprietário o criador do BitKeeper queria ficar próximo da comunidade e participava no desenvolvimento do kernel Linux , é tanto que foi lançado uma versão community;
- O maior ponto positivo dele é por conta de não parecer nem um pouco com o CVS;
- Por conta de ter uma versão da comunidade , o maior cliente do BitKeeper era o Linux , para ser mais exato , os desenvolvedores do Kernel Linux;

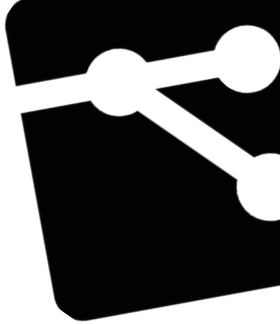
# 4. O que é o Git ?

- Git é nada mais nada menos que um versionador de arquivos sendo escrito em C , Shell e Perl ,
- Utilizado por um ou mais desenvolvedores em um projeto;
- Através do Git é possivel fazer contribuição no projeto de modo prático e simples e cabendo apenas ao criador/dono ou contribuidores que tem permissão do projeto a incluir as alterações.

# 5. Como surgiu o Git ? Um pouco sobre sua história e desenvolvimento.

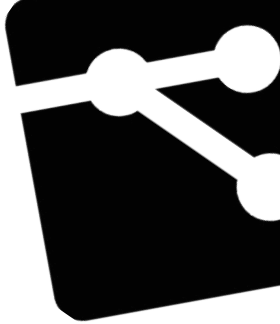
- Desenvolvimento do Git está entrelaçado com o kernel Linux, já que por sua vez começou após vários desenvolvedores do kernel decidirem desistir de acessar ao sistema do BitKeeper que era um software proprietário. Isso por conta que o BitKeeper deixou de ser gratuito por conta de direitos autorais , pois foi feita uma acusação dizendo que tinha sido realizado uma engenharia reversa nos protocolos do BitKeeper , porém , foi demonstrado depois em uma apresentação na Linux.Conf.Au , que o processo de engenharia reversa teria sido utilizado para direcionar um telnet para a porta apropriada de um servidor BitKeeper e digitar o comando "help" para se obter informações de ajudar.

# 5. Como surgiu o Git ? Um pouco sobre sua história e desenvolvimento.



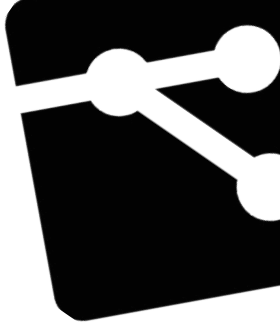
- Com isso , Torvalds criador do Linux queria um sistema versionador de arquivos para que ele pudesse usar de forma similar ao BitKeeper , porém Torvalds não encontrava nenhum parecido e ele entrou em ação desenvolvendo o Git , o desenvolvimento se iniciou no dia 3 de abril de 2005 , foi anunciado no dia 6 de abril e foi lançado se tornando auto-hospedeiro em 7 de abril.

# 5. Como surgiu o Git ? Um pouco sobre sua história e desenvolvimento.



- Durante o desenvolvimento do Git , Torvalds teve vários critérios para o projeto , sendo eles :
  1 – Tomar o CVS como um exemplo do que não fazer;
  2 – Suportar um fluxo distribuído , como o do BitKeeper;
  3 – Proteção contra o corrompimento de arquivos;
  4 – Alto desempenho.

# 5.1. O que significa a sigla Git

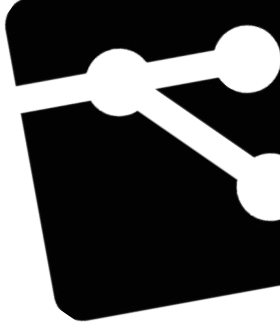


- Quando perguntaram sobre o porque do nome para Torvalds , Torvalds satirizou sobre o termo “Git” , uma gíria em inglês britânico para cabeça dura.
- Uns dizem que Git significa Global Information Tracker ( Rastreador de Informações Globais ).
- Porém o verdadeiro significado é Goddamm Idiotic Truckload of shit. ( Maldito caminhão idiota cheio de merda )

# 6. Como o Git pode ajudar no desenvolvimento de um código.

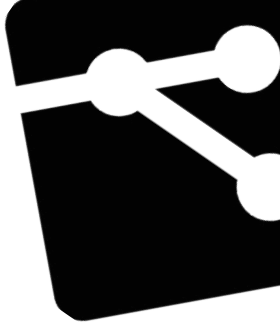
- Como foi citado , o Git é um sistema versionador de arquivos , ou seja , ele vai ter um histórico de versões do código , vai ter a última modificação realizada no código , assim por diante , com isso o Git se torna muito útil no desenvolvimento ou até mesmo na finalização de um código nos seguintes casos :
- Evitar criar vários arquivos como "versão final" , exemplo : Codigo_final1 , Codigo_final2 , Codigo_final3 e assim por diante;
- Evita a perca total do código em casos de corrompimento do arquivo de boa ou má intenção;
- Acessar versões antigas do código quando alguma função do código não estava bugada;
- Evita conflitos repetidos de mesclagem de código;
- Evitar envio de códigos através de e-mails , pendrives , CD/DVDs ou pombo correio.

# 7. Vantagens de usar o Git.



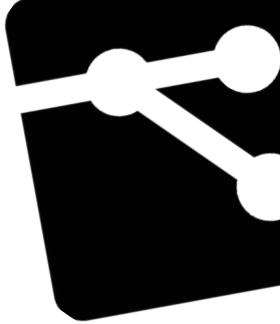
- Não há necessidade de conectar a um servidor central;
- Pode trabalhar sem uma conexão com a internet;
- Os desenvolvedores podem trabalhar de forma independente e mesclar seu trabalho mais tarde;
- Cada cópia de um repositório Git pode servir como servidor ou como cliente além de ter o histórico completo.

# 7. Vantagens de usar o Git.



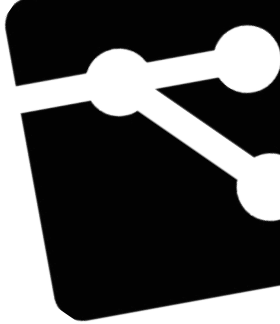
- Apesar do Git ter muitas vantagens , ele tem apenas 1 desvantagem que deixa ele ser “ruim” : necessidade de maior conhecimento da ferramenta para poder usar.

# 8. Como realizar a instalação do Git na sua máquina.



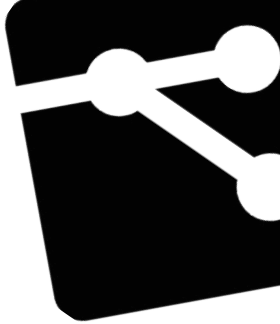
- Nem todas as maquinas possuem Git instalado de fabrica , para verificar se sua maquina possui Git , abra o Terminal dependendo do seu S.O e digite : git —version , assim vai estar aparecendo a versão do Git instalada na sua maquina.
- O Git pode ser instalado em máquinas que possuem os seguintes sistemas : macOS , Windows e distribuições Linux/Unix.
- A instalação pode estar sendo realizada acessando o site oficial do projeto Git : https://git-scm.com/ e selecionar a opção Downloads.
- Acessando o link selecione qual S.O está instalado na máquina e siga os passos a seguir para cada S.O :

# 8.1. Instalando o Git no Windows.



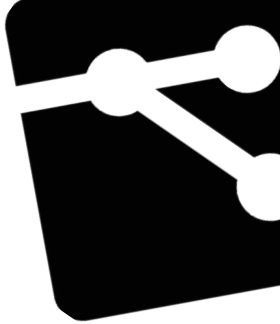
- Selecionando a versão Windows , selecione em seguida a arquitetura da máquina , se é 32 bits ou 64 bits.
- Depois de baixar o executável de acordo com suas configurações , é só clicar em Next/Próximo , ver se você quer mudar alguma configuração , como por exemplo o local de instalação do Git , caso você queira mudar. Isso se for a versão Standalone Installer , a versão Portable é mais simples de se instalar apertando somente o botão install/instalar.
- Pode ser instalado também através também do winget ( Windows Package Manager ) pelo powershell.

# 8.2. Instalando o Git no macOS.



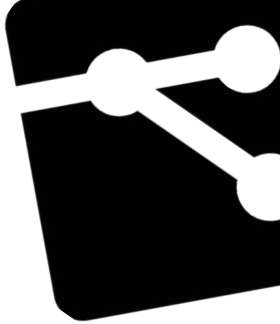
- No Mac já tem uma chance de ter o Git instalado por padrão no sistema , para checar é só dar o comando : git —version , assim você vai obter a versão do Git.
- Caso o contrário , selecione a opção de instalação macOS , pode está sendo realizado através dos seguintes gerenciadores de pacotes : Homebrew ou MacPorts.
- Pode também está sendo realizado através de outras maneiras , como por exemplo : Xcode o ambiente de desenvolvimento integrado e software livre da Apple , fazendo a instalação através da fonte direto ou até mesmo estar instalando uma versão GUI do mesmo ou instalação através de pacotes binários.
- Todas as formas de instalação do Git no macOS se encontram disponíveis no site do projeto.

# 8.3. Instalando o Git em distribuições Linux/Unix.



- Selecionando a opção de instalação Linux/Unix, pode está sendo realizado em várias distribuições Linux , algumas distribuições já vem com o Git instalado de padrão.
- Em algumas distribuições que não aparecem a linha de código de instalação do Git , é por conta que provavelmente para instalar é através de pacotes que a própria distribuição disponibiliza , exemplo : Slackware.

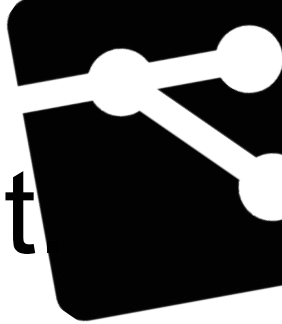
# 9. Configurando o Git associando seus dados.



- Para configurar o Git inicialmente em sua máquina , execute os seguintes comandos :

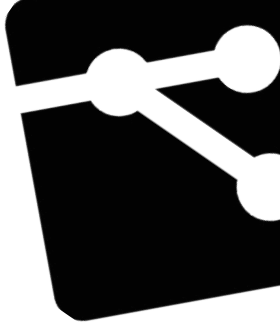
```
git config —global user.name "SEU NOME"
git config —global user.email "seuemail@dominio.com"
```

# 10. Termos que são muito usados no Git



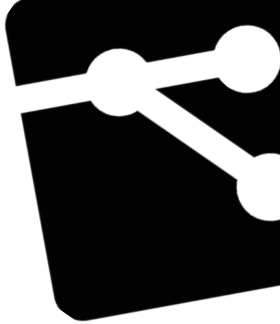
- Repository  : Repositório , local onde fica todos os arquivos do projeto , incluindo os históricos e versões.
- Commit : Enviar , Coleção de alterações realizadas, podemos definir esse termo como um backup do seu projeto, sempre que necessário você pode retrocede até algum commit.
- Branch : Ramo , é uma ramificação do projeto , cada branch representa uma versão do projeto , e podemos seguir uma linha de desenvolvimento a partir de cada branch.
- Fork : Bifurcação , a sua tradução já diz tudo , é basicamente uma cópia de um projeto existente para seguir uma nova direção.
- Merge : Mesclar , capacidade de incorporar alterações no git , onde acontece uma junção dos branchs.
- Head : Referência do commit atual / última versão

# 11. Lista de linguagens mais usadas no Git.



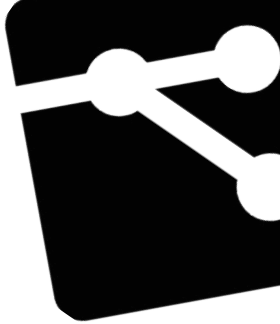
- HTML;
- CSS;
- Javascript;
- Python;
- ASP;
- Scala;
- Shell scripts;
- PHP;
- Ruby;
- Ruby on Rails;
- Perl;
- Java;
- C/C++;
- C#;
- Objective C;
- Haskell;
- CoffeeScript;
- ActionScript;
- Kotlin;
- Nodejs;
- MySQL.

# Observação !



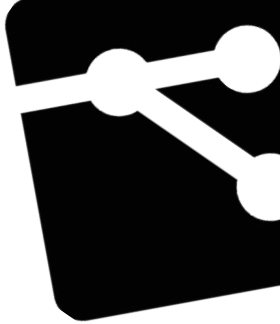
- Apesar de ter muitos códigos e tal , o Git também aceita outros tipos arquivos como .pdf , .psd , .jpg , .png etc , então é possível você encontrar materiais como livros digitais na plataforma.

# 12. Lista de projetos que utilizam Git.



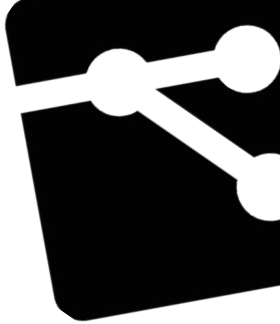
- Android;
- Debian;
- Arch Linux;
- Btrfs;
- Clojure;
- Eclipse;
- Fedora;
- GIMP;
- Kernel Linux;
- PHP;
- Xfce;
- Perl;
- Linux Mint;
- Qt;
- Ruby on Rails;
- VLC;
- Wine;
- GNOME;
- COSMIC;
- Reddit;
- Kernel Linux;
- jQuery.

# 13. Começando a mexer no Git.



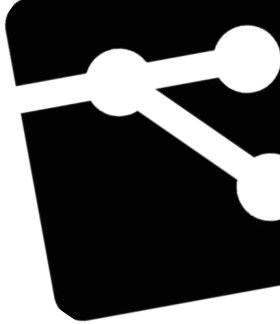
- Agora vamos ver alguns comandos que tem disponíveis no Git e os que são importantes realmente na hora do desenvolvimento de seu projeto , você deve estar se perguntando o porque precisa dos comandos Git e tal.
- Como foi citado em tópicos anteriores, uma das vantagens do Git é o trabalho paralelo entre pessoas, isso porque o Git evita que o código não vire uma bagunça para que isso seja possível e assim podendo também criar varias ramificações ( branches ).
- Outro motivo que também foi citado , é por conta de voltar em alguma versão antiga do código.

# 13.1. Comandos do Git.



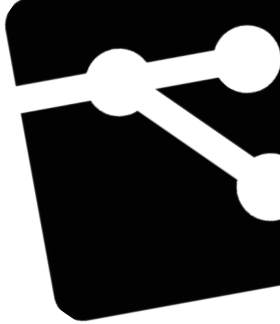
- Agora pouco vimos um dos comandos do Git , que foi o de configuração inicial , configuramos nada mais nada menos o nosso nome e e-mail para identificação nos commits.
- Aqueles tipos de comando são classificados como comandos de configuração( Git config ), tem outros comandos de configuração , sendo eles : git config —global core.editor vim , git config —global merge.tool vimdiff , git config —global core.excludesfiles ~/.gitignore e git config —list.
- Para usar os comandos do Git , você primeiro abre o seu terminal de acordo com seu sistema.

# 13.1. Comandos do Git.



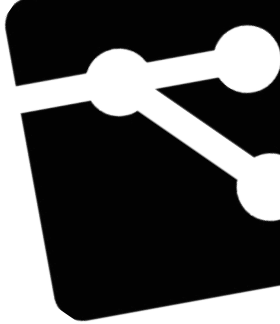
- Para começarmos as praticas dos comandos disponíveis no Git , por que não iniciar com comandos de repositório, primeiramente , vamos criar nosso repositório.
- Para isso, primeiramente vamos criar uma pasta referente ao nosso repositório , após criar a pasta , vamos abrir a mesma e dar o comando git init para estar transformando a pasta em um repositório Git.

# 13.1. Comandos do Git.



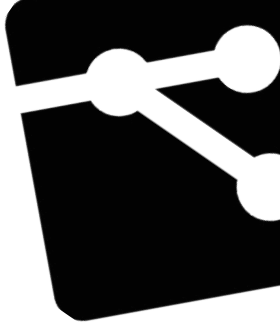
- Depois de inicializar o repositório , uma coisa para se fazer é mudar a nossa branch de master para main , sempre que inicializamos um repositório estamos criando ele como master e de um tempo pra cá eles estão migrando de master para main , não é nada de absurdo ou coisa do tipo , é só questão de nomenclatura mesmo. Não é obrigado a mudar nem nada , é apenas uma boa prática para se fazer.
- Para estar realizando isso é só dar o comando : git branch –M "main"
- Tem como deixar isso automático digitando : git config —global init.defaultBranch “main” , mas depois de dar esse comando ainda precisa dar o git branch –M “main”

# 13.1. Comandos do Git.



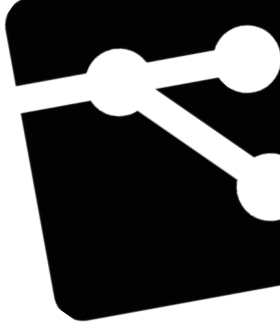
- Bom , já que criamos nosso repositório , vamos adicionar arquivos a este repositório, então vamos criar algum arquivo para este repositório local.
- Depois de criar , vamos agora adicionar esse arquivo para ser enviado na próxima vez que ocorrer um commit , através do : git add <nome_do_arquivo>
- Mas caso quiséssemos que todos os arquivos fossem adicionados em casos específicos , poderia ser dado o comando git add das seguintes formas : git add . , git add –a , git add *.
- Para realizar o processo inverso , ou seja , remover do add , só realizar o comando git restore <nome_do_arquivo> ou git restore para tudo.

# 13.1. Comandos do Git.



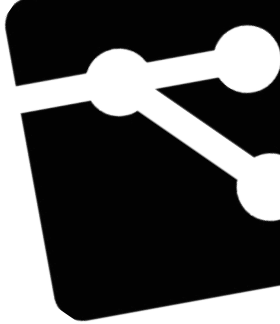
- Agora iremos realizar nosso primeiro commit , para estar realizando o commit vamos estar utilizando o comando git commit.
- Porém precisamos passar alguns parâmetros extras para estar realizando nosso commit , sendo eles o : –a ( Esse parâmetro informa que todos os arquivos alterados devem ser incluídos no commit ) e o –m ( para adicionar uma mensagem explicativa para o commit ). Sendo assim , vamos digitar da seguinte forma : git commit –a –m "mensagem da commit".

# 13.1. Comandos do Git.



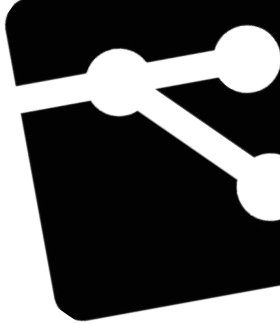
- O git é inteligente o suficiente para realizar apenas commits se realmente tiver alguma alteração nos arquivos , então se você der vários git add e o arquivo só tiver uma única modificação , ele só vai ser commitado 1 vez.

# 13.1. Comandos do Git.



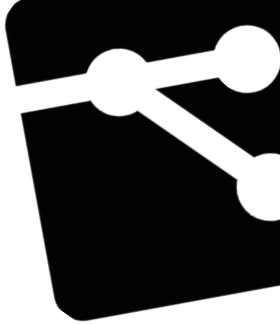
- E agora se você quiser checar como ficou seu commit , é só dar agora o comando git log e ver todos os commits realizados no seu repositório local.
- E caso o arquivo seja modificado e der o comando git status , vai aparecer que o arquivo sofreu modificações e que você pode dar commit.
- E com o comando git show , mostra as alterações realizadas no último commit.

# 13.1. Comandos do Git.



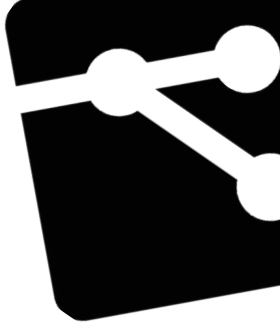
- Como já foi citado , sabemos que os repositórios Git possuem suas branches , para checarmos as branches do nosso recém-nascido repositório é só dar o comando : git branch. Assim ele vai estar retornando todas as branches e a branch que estamos trabalhando atualmente que é simbolizada com um "*" antes do nome e a letra é colorida , como criamos ele recentemente , vai retornar para nós apenas a seguinte branch : main.
- Para estar sendo criado uma nova branch , não tem muito mistério , é só dar o comando : git branch <nome_da_branch>
- Podemos criar branches com bases outras branches , da seguinte maneira : git branch <nome_da_branch> <branch_ja_existente>

# 13.1. Comandos do Git.



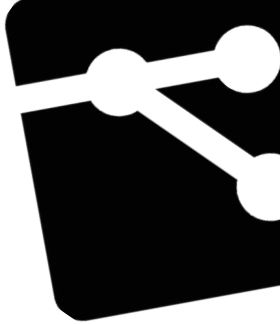
- Um termo que é consideravelmente bem usado é o "branch corrente", esse termo está se referindo ao branch que está sendo trabalhado no momento.
- Criamos as novas branches e tudo mais , porém a branch corrente ainda continua sendo o main , não mudamos de branch ainda , para realizarmos tal proeza , é só executar o comando : git checkout "nome_da_branch" , e uma forma relativamente rápida de criar uma branch nova e já trocar de branch é através da : git checkout –b "nome_da_nova_branch".
- E para deletar uma branch , é só dar o comando : git branch –d nome_da_branch. Não é possivel deletar uma branch corrente , ou seja , não é possivel deletar uma branch que esteja selecionada e sendo trabalhada no momento.

# 13.1. Comandos do Git.



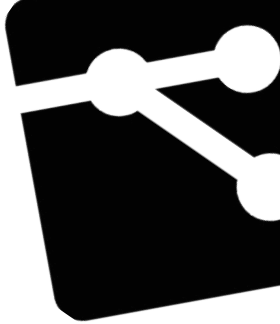
- Agora vamos supor a seguinte situação , como fariamos para o nosso arquivo sofrer alterações em outra branch ser em outra branch ? E como fariamos para que essa alteração chegasse até a branch main ?

# 13.1. Comandos do Git.



- Bom , para dar commit em outra branch não tem mistério , só dar o comando git checkout e o nome da branch e aplicar as modificações por lá.
- Agora para fazer chegar até o nosso main já é outro processo. Como já foi citado , temos um termo chamado Merge que serve para mesclar , é aqui onde o merge entra em ação , bom , primeiramente vamos nada mais nada menos que git checkout main e vamos finalmente usar o comando de mesclagem que é : git merge branch_que_deseja_mesclar.

# 13.1. Comandos do Git.



- Então a ordem de comandos para realizar a mesclagem da situação citada seria :

Supondo que o nome da branch originada da main fosse developer

git branch main developer

git checkout developer

Obs : Uma alternativa inves de dar 2 linhas de comando , poderia ser dado também git checkout –b developer , assim criaria a branch e ja trocaria logo de branch.

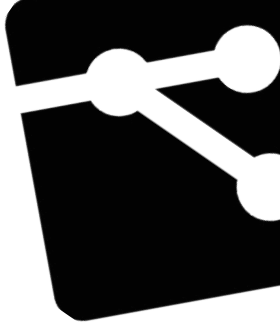
git add .

git commit –a –m “Comentario”

git checkout main

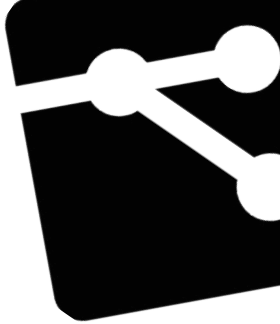
git merge developer

# 13.1. Comandos do Git.



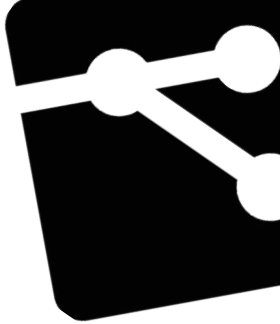
- Agora vamos supor a outra situação , vamos supor que precisássemos fazer um merge em um branch main que já tivesse alguma alteração depois que uma outra branch originada dele foi criada, com isso , o branch originado dele não estava refletindo a última versão do main , então como podemos estar procedendo com esse problema ?

# 13.1. Comandos do Git.



- Nesse caso vamos nada mais nada menos que fazer a rebase da branch originada da main. Rebase é o comando do git que pega o ultimo commit do branch main e traz para outro branch e aplica todas as suas commits.
- A solução do problema poderia ser fazendo somente com merge , porém , tem chances de causar conflitos nos arquivos e corrompe-los.

# 13.1. Comandos do Git.



- Sendo assim a ordem dos comandos ( supondo que a branch que vai sofrer rebase se chame developer ):

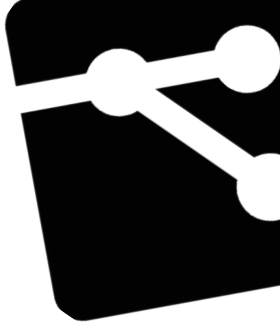
  git add .

  git commit –a –m

  git checkout developer

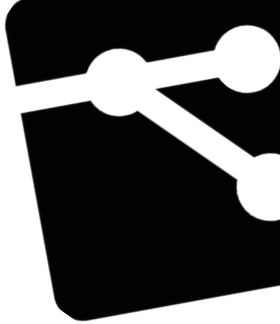
  git rebase main ( precisa especificar qual branch ele vai sofrer a rebase )

# 13.1. Comandos do Git.



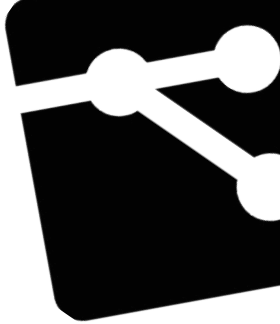
- E agora como estar voltando para uma versão anterior do nosso repositório local.
- Sabemos que o git log aparece todas as modificações que aconteceram.
- Porém para gente saber o hash certinho da commit e retornar para uma versão antiga , usaremos —oneline para dar informações resumidas das commits em uma linha.
- Bom , para isso vamos executar o comando git log —oneline.
- E em seguida pegaremos o hash e vamos dar o comand git checkout <hashcode>.

# 13.1. Comandos do Git.



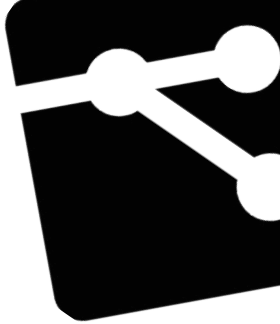
- Você deve estar se perguntando , tem como excluir commits ?
- A resposta é sim , pra ser mais exato , tem como reverter commits.
- Você deve ta se perguntando mais ainda , como ? Bom , vou mostrar 2 comandos que pode estar sendo utilizado para realizar isso.

# 13.1. Comandos do Git.



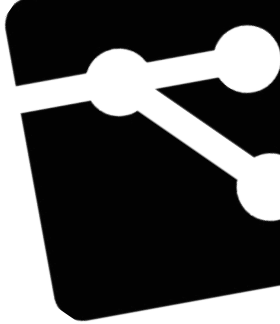
- O primeiro comando é o git revert <hash do commit> , esse comando ele fará um novo commit no projeto revertendo todas as mudanças até o commit indicado.
- O segundo é o git reset <hash do commit> , esse comando ele vai retornar toda a árvore das commits até a commit indicada , as mudanças que você deseja realizar vão estar como “not staged” , mas caso você não queira fazer nenhuma mudança , pode estar adicionando o —hard e assim vai apagar até as mudanças.

# 13.1. Comandos do Git.



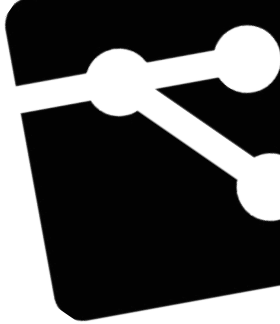
- Sabemos que temos o git show e o git log para apresentar o historico de commits , e junto deles temos o git diff que mostra apenas o que mudou com os commits.
- Outro comando bastante útil é o git rm , com esse comando podemos estar excluindo uma pasta ou arquivos do nosso repositório , é só dar o comando : git rm <nome_do_arquivo_ou_pasta>.

# 13.1. Comandos do Git.



- Por último , vamos falar do comando : git stash;
- Podemos dizer que o git stash é a fase de testes de seu arquivo modificado;
- Para exemplificar melhor , vamos supor que temos um commit pendente , caso você use o git stash , o Git vai criar uma branch temporária contendo a versão temporária, isso se pelo menos já tiver tido 1 commit pelo menos no repositório;

# 13.1. Comandos do Git.



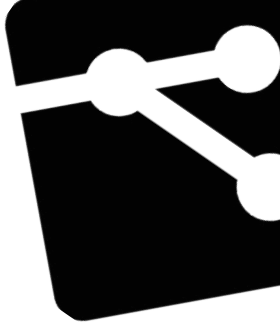
- O git stash possui os seguintes comando :

  git stash show -> lista os arquivos modificados no último stash;

  git  stash clear -> realiza a remoção de todos os stashs;

  git stash save <mensagem> -> Coloca uma mensagem no stash

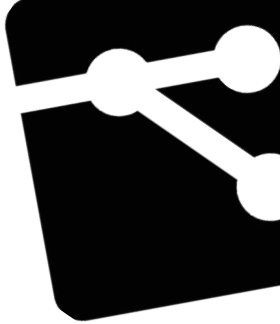
  git stash pop -> realiza a remoção em ordem de pilha.



# 14. Finalizando sobre Git.

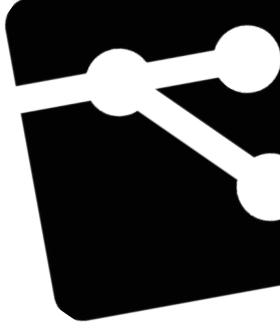
- Vimos muita coisa sobre Git , o que ele é , como surgiu , seu desenvolvimento , como funciona , seus principais comandos etc. Agora então vamos botar o conhecimento obtido de Git em prática criando repositórios locais para ter um conhecimento bom e finalmente partir para Github e usar repositórios remotos.
- Então bora para as questões praticar um pouco.

# 14. Finalizando sobre Git.



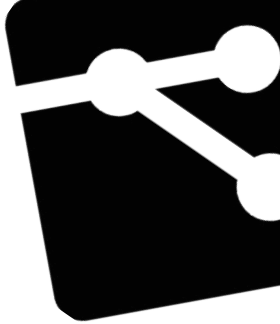
- Exercício 1 : Criar uma pasta com um nome qualquer para seu repositório da sua escolha , inicializar o git e criar um arquivo .txt. Depois disso adicionar o seguinte conteúdo para o arquivo : "SECOMP2022". Em seguida realizar o commit com a seguinte mensagem : "Commit 1" , e depois exibir o registro de commits.

# 14. Finalizando sobre Git.



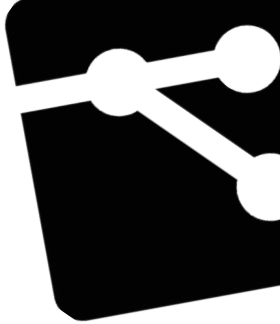
- Exercício 2 : Criar uma nova branch e nessa nova branch mudar o conteúdo do arquivo colocando apenas um “2” na frente da SECOMP2022 e dar commit com a mensagem “Commit 2” e subir as modificações para a branch main.

# 14. Finalizando sobre Git.



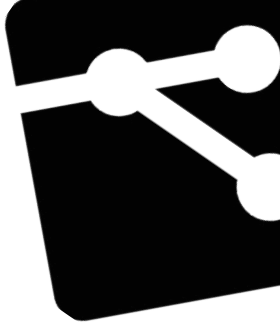
- Exercício 3 : Modifique o arquivo txt na branch main, adicionar , e dar commit e logo em seguida fazer a branch recém criada ter as alterações da branch main.

# 14. Finalizando sobre Git.



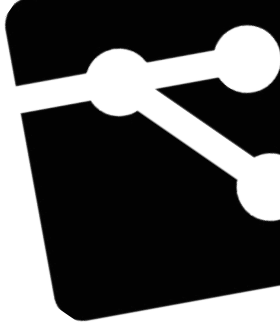
- Exercício 4 : Faça uma modificação no arquivo .txt na branch main e logo em seguida coloque na fase de testes, confirme se foi adicionado e depois exclua.( Tudo isso na branch main ).

# 14. Finalizando sobre Git.



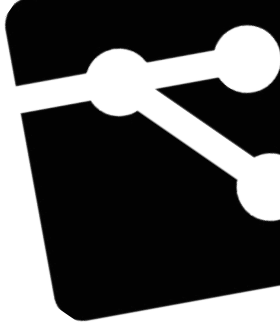
- Exercício 5 : Faça a mesma coisa da questão anterior , só que colocando uma mensagem/nome de “SECOMP”.

# 14. Finalizando sobre Git.



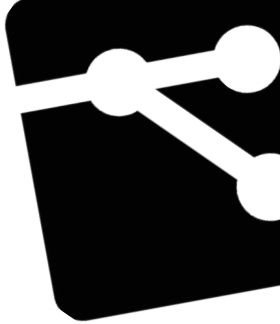
- Exercício 6 : Volte para uma versão antiga do commit na branch main perdendo todas as alterações.

# 14. Finalizando sobre Git.



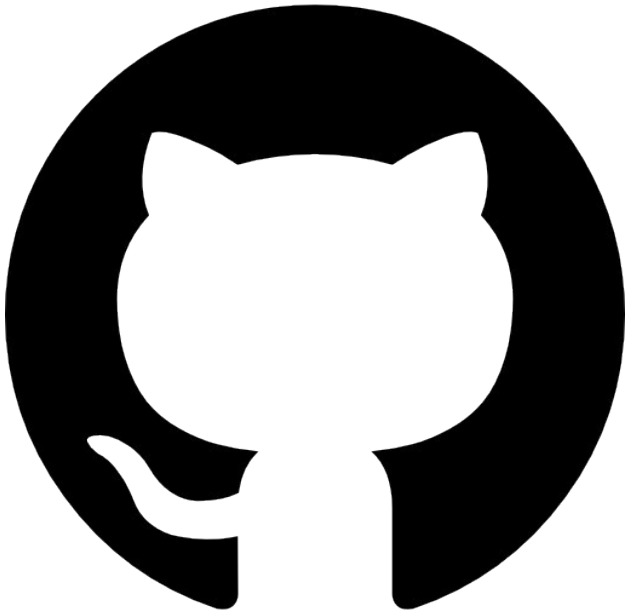
- Pergunta 1 : Para que serve o Git config ?
- Pergunta 2 : Para que serve o Git add ?
- Pergunta 3 : Qual o diferencial do Git em relação ao seus antecessores ?
- Pergunta 4 : Qual foi o fator que desencadeou a criação do Git ?
- Pergunta 5 : Git e Github é a mesma coisa ?
- Pergunta 6 : O que é o git stash ? De acordo com o que foi abordado
- Pergunta 7 : Se for realizado 2 stashs , primeiro stash com o código 432bcef , segundo 156adgh e o terceiro 890jilk respectivamente , se for realizado o comando git stash pop , qual vai ser o stash removido primeiro e por último ?

# 14. Finalizando sobre Git.



- Pergunta 8 : É possível deleta uma branch que esta em uso ?
- Pergunta 9 : É possível criar uma branch com base uma branch que foi criada a partir da main ?
- Pergunta 10 : O Git suporta tipos de arquivo além das linguagens de programação ? Exemplo : .pdf , .psd , .jpg etc.

# Github



# 15. O que é o Github.

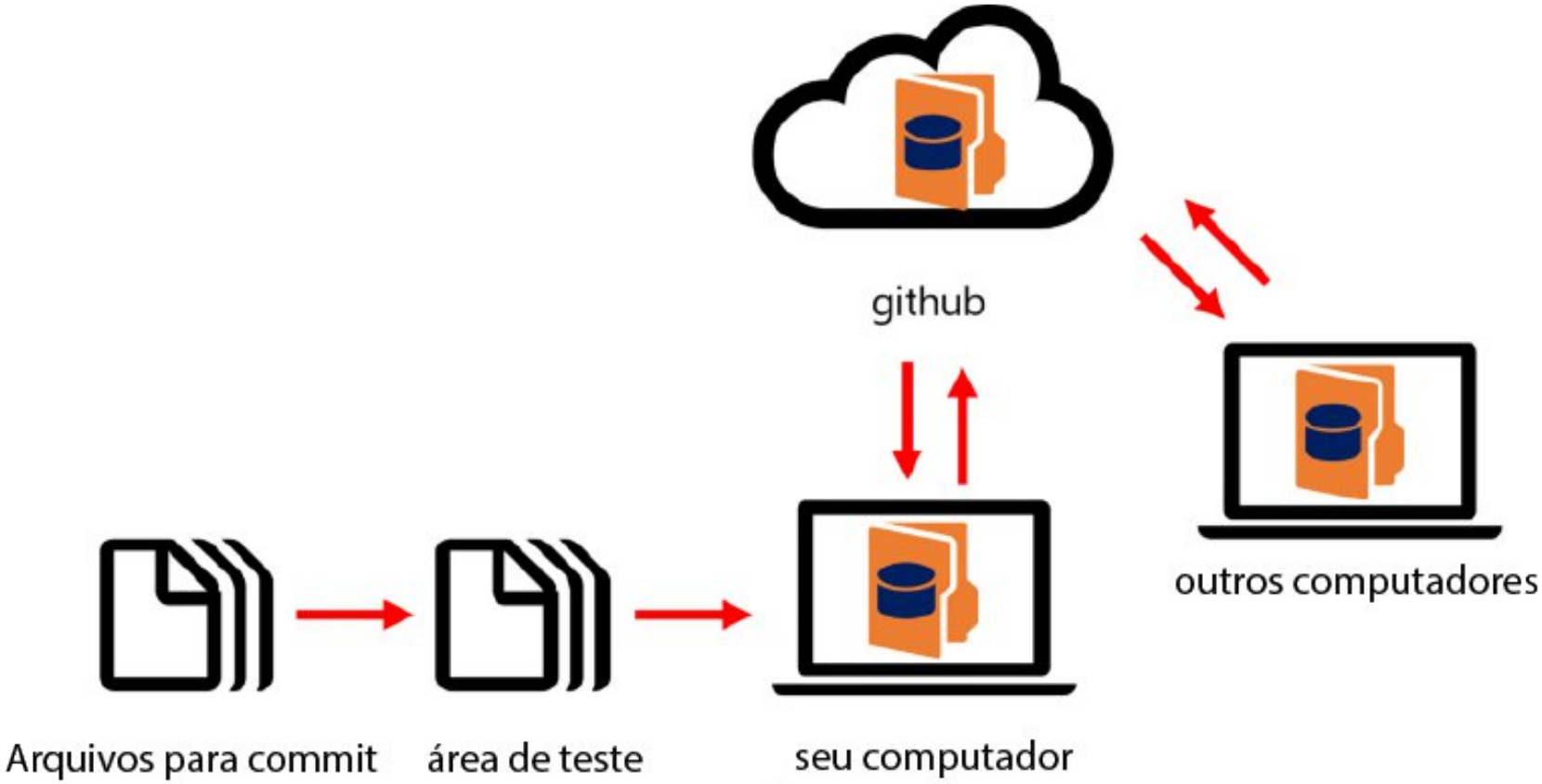
- Como foi exibido , Git é nada mais nada menos que o sistema versionador de arquivos, funcionando apenas de maneira local , nada remotamente , equanto isso , o Github é a plataforma de hospedagem para o Git , ou em outras palavras , um repositório remoto.
- Lembrando , Git e Github são coisas totalmente diferente , enquanto Git é o sistema versionador , Github é a plataforma do Git para ser possível criar repositórios remotos.
- Como já foi citado mais cedo , como o Git aceita vários tipos de arquivos como .pdf , .psd etc , você provavelmente encontrará muito material no Github também de estudos , como por exemplo : livros digitais.

# 16. O que são os repositórios remotos.

- Repositórios remotos é simplesmente um serviço na nuvem onde você salva ou publica o projeto que pode ser sim ou não público para outras pessoas.
- Github não é o único repositório remoto que existe , porém é um dos mais utilizados pelo mundo , tanto que se você perguntar pra algumas pessoas onde ela publica os códigos dela , a grande maioria vai responder Github.

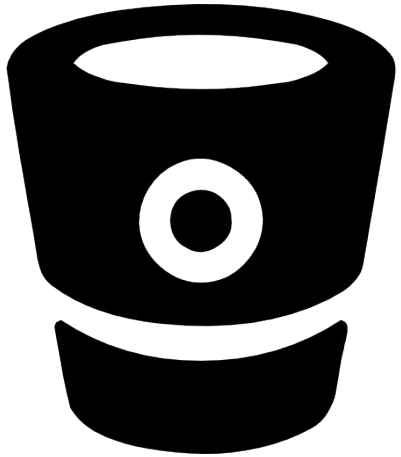
# 17. Como funciona um repositório remoto.

- De forma resumida , um repositório remoto funciona da seguinte maneira :



# 18. Outros repositórios remotos.

- Como foi citado agora pouco , existe outros repositórios remotos existentes , e vou citar algum deles aqui, talvez um deles você já viu ou tenha ouvido falar sobre :





# 19. Outros repositórios remotos.

- Existe vários outros repositórios remotos como : Google Code , SourceRepo , Gitorious , Azure DevOps etc , porém os que mais se destacam são os citados agora pouco em imagens.

# 20. A história do Github e seu desenvolvimento.

- Github foi desenvolvido por 4 pessoas , sendo eles : Chris Wanstrath , J. Hyett , Tom Preston-Werner e Scott Chacon
- Foi criado usando a framework Ruby On Rails e seu desenvolvimento começou em fevereiro de 2008 , anos depois do lançamento do Git.
- O principal objetivo do Github é abrigar projetos que são visionados via Git.
- Nos primeiros 5 meses de 2011 , teve 1,1 milhões de commits registrados ( o SourceForge que na época era o melhor , no mesmo período ficou em segundo lugar com 600 mil commits ).
- Não demorou muito para que o Github ficasse mais popular que o SourceForge.
- Com o passar do tempo , o Github estava se tornando bastante popular, e com isso foi possivel perceber a necessidade de implementações de funcionalidades que o deixassem com uma cara de rede social.

# 21. A história do Github e seu desenvolvimento.

- Cerca de 2 anos depois do seu lançamento o Github estava tendo marcos incríveis , tanto de usuários registrados quanto de repositórios remotos criados no site.
- Em 2018 , pra ser mais exato , 4 de junho , a Microsoft anuncia que havia chegado a um acordo para adquirir o Github por 7,5 bilhões de dólares , a compra foi encerrada no dia 26 de outubro do mesmo ano.
- Durante esse meio tempo enquanto rolava a compra da Microsoft , Github expandiu o Github Education , oferecendo pacotes de educação gratuitos para todas as escolas.

# 21. A história do Github e seu desenvolvimento.

- O motivo da Microsoft comprar o Github foi pra concentrar seus esforços nas quatro principais frentes que era crucial : Nuvem , IA , serviços ( Windows , Office e Xbox entre outros ) e mercado corporativo. Além da Microsoft ser a companhia privada que mais colaborava com o Github.
- Github mesmo sendo comprado pela Microsoft não alterou nada na forma como opera.

# 21. A história do Github e seu desenvolvimento.

- Agora vamos saber sobre a origem do mascote do Github , muitos se perguntam o porque que o mascote do Github é um gato polvo ou também conhecido como Octocat.
- Não é nenhuma história surpreendente , na realidade é uma história bem besta.
- O motivo do mascote do Github ser o Octocat é que Tom Preston-Werner estava procurando uma imagem engraçada para a página 404 do Github e encontrou o Octocat no iStockphoto ( Sim , o site que vende imagens ) e comprou a licença da imagem de uso por apenas 50 doláres.

# 21. A história do Github e seu desenvolvimento.

- Outras pessoas preferem outra versão dessa história por conta dela ser meio besta , que no caso é a seguinte : Como Git tem ramificações , tecnicamente os tentáculos de um polvo também é suas ramificações e tinha o fato do Tom gostar muito de gatos , sendo assim surgindo o Octocat.
- Em particular eu prefiro a história original que eles compraram os direitos da imagem.

# 22. Alguns projetos famosos que é possivel encontrar no Github.

- OBS Studio;
- Android;
- VSCode;
- React Native;
- COSMIC;
- NixOS

# 23. Empresas que utilizam Github.

- Twitter;
- Facebook;
- Yahoo;
- Mozilla;
- Microsoft;
- Redhat.

# 24. Vantagens do Github.

- Portfólio;
- Comunidade;
- Trabalho em equipe;
- Controle de incidentes/bugs;
- Contas comerciais e gratuitas.

# 25. Configurando uma chave SSH para o Github.

- Antes de tudo , você deve estar de perguntando , o que seria essa chave SSH ( Secure Socket Shell ). Uma chave SSH é nada mais nada menos um componente do protocolo Secure Shell que garante a proteção dos sistemas de TI durante a transferência de dados.
- Agora você deve estar se perguntando , o porque raios você deve criar uma chave SSH , além é claro de garantir segurança como descrito anteriormente , também garante acessar e gravar dados em repositórios do Github. Pois ao você se conectar por meio do SSH , você autentica usando um arquivo de chave privada no computador local.

# 26. Configurando uma chave SSH para o Github.

- A criação da chave SSH vai estar sendo realizada ao vivo no minicurso , porém pode estar sendo consultado também pelo proprio site do Github sobre documentação , que inclusive , vai ser usado no passo a passo no minicurso para tentar explicar cada detalhe.
- Site sobre a documentação do Github : https://docs.github.com/

# 27. Explorando algumas configurações do Github.

- Vamos ver algumas opções disponíveis no Github tanto de conta quanto de repositório.

# 28. Criando repositórios remotos.

- Até o momentos só criamos repositórios locais , esses repositórios locais só podem ser acessados caso estejam conectados no mesmo servidor.
- Vamos supor a seguinte situação , você está em um outro ambiente de trabalho , onde o seu repositório local não pode ser acessado ( a não ser que você tenha salvo no email ) , isso seria um baita de um problema.
- Os repositórios remotos visam resolver esse problema , sendo possível acessar o seu repositório de qualquer lugar e estar clonando e continuando desenvolvendo o projeto.

# 28. Criando repositórios remotos.

- Antes de entrar em ação realmente com os repositórios remotos , vamos aprender só alguns comandos extras do Git para realmente entrar em ação.
- Não são comandos complexos e vão ser importantíssimos para estarmos mexendo com nossos repositórios remotos.

# 28. Criando repositórios remotos.

- git remote , esse comando estabelece a conexão com o repositório local e um remoto , vamos ver ele daqui a pouco quando formos criar o repositório remoto no github.
- git push , esse comando serve para subir as modificações , ou “empurrar” elas para o repositório remoto , é claro , que pra poder usar ela precisa ter usado o git remote anteriormente.
- git fetch , esse comando é essencial para quem trabalha com outros membros em um projeto , e para saber todas as informações dos commits você pode estar dando simplesmente : git fetch. O motivo dele ser essencial é por conta dele ser um comando conjunto do git remote , branch , checkout e reset, assim baixando os conteúdos disponíveis em um repositório remoto.

# 28. Criando repositórios remotos.

- git pull , do mesmo jeito que o push “empurra” as modificações para o repositório remoto , o git pull ele puxa as informações/atualizações que ocorreram no repositório , para estar utilizando é só dar o comando : git pull.
- git clone , esse comando é muito essencial para aqueles que querem clonar um repositório para ter uma base para o seu projeto , ou até mesmo começar a colaborar em algum projeto sendo por uma branch ou fork , para estar sendo utilizado o comando é só dar : git clone <url_do_repositório>.

# 28. Criando repositórios remotos.

- Bom , agora vamos criar o nosso repositório remoto no Github e aprender como subir o nosso repositório local para o repositório remoto de um jeito simples e fácil.
- De brinde vamos criar um README.md sobre nós mesmo para deixar mais chamativo ainda o perfil Github.

# 29. Praticando com exercícios.

- Finalizamos a parte de Github , então vamos botar em pratica o que vimos até o momento com alguns exercícios de fixação , não muito difíceis.
- Exercício 1 : Crie uma conta no Github , faça sua chave SSH e coloque na sua conta Github ( Recomendo fazer isso em uma guia anonima ).
- Exercício 2 : Com o repositório local criado na parte de Git , faça um repositório remoto e suba tudo do repositório local.
- Exercício 3 : Faça alterações no arquivo .txt criado no repositório local e suba essas modificações para o Github.

# 29. Praticando com exercícios.

- Exercício 5 : Esse exercício é para ver como funciona realmente uma Fork e um PR , então vou criar um repositório e quero que vocês adicionem arquivos e façam um PR.

# 29. Praticando com exercícios.

- Pergunta 1 : O que é um Pull Request ?
- Pergunta 2 : Diferença entre fork e uma branch ?
- Pergunta 3 : O que é o Github ?
- Pergunta 4 : Para que serve o git pull ?
- Pergunta 5 : Qual diferença entre git fetch e o git pull ?
- Pergunta 6 : É possível criar uma fork de outra fork ?
- Pergunta 7 : Existe somente o Github como plataforma de hospedagem de repositório remoto ?
- Pergunta 8 : Em qual framework foi criado o Github ?
- Pergunta 9 : Para que serve a opção Issues ?
- Pergunta 10 : O que é Github Actions ?

# 30. Finalizando sobre Github.

- Bom , finalmente foi finalizada a parte do Github , a parte do Github foi menor que a do Git por conta que o Github é apenas um complemento por ser um repositório remoto , mas não deixa de ser importante é claro.
- Git e Github tem muito a ser explorado ainda, o que foi apresentado nesse minicurso foram apenas conceitos básicos para intermediário, em resumo , foi pra dar uma luz pra quem quer começar a mexer com Git. Caso tenha interesse de realmente se aprofundar nisso , tem disponível no proprio site do Github sobre a documentação de usar Git para gerenciar repositórios do Github e no próprio site do Git tem uma documentação muito mais detalhada e consideravelmente um pouco mais complexa sobre seus comandos.

# 30. Finalizando sobre Github.

- Site da documentação do Git :

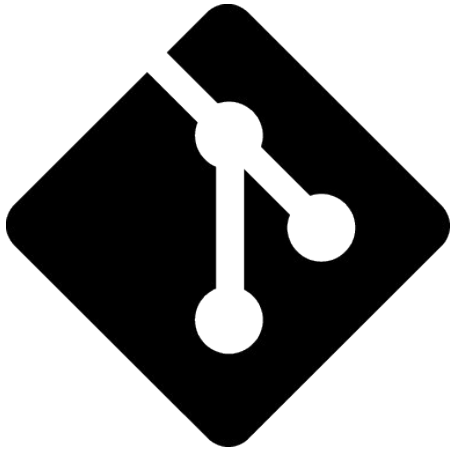
  https://git-scm.com/docs/git/pt_BR

- Site da documentação do Git para gerenciar repositórios do Github :

  https://docs.github.com/pt/get-started/using-git

# 30. Finalizando sobre Github.

- O slide vai estar sendo disponibilizado no e-mail de cada um de vocês que participaram do minicurso , caso queira estar consultando alguma parte em especifico.



# Obrigado pela atenção.