



HELLO
WORLD



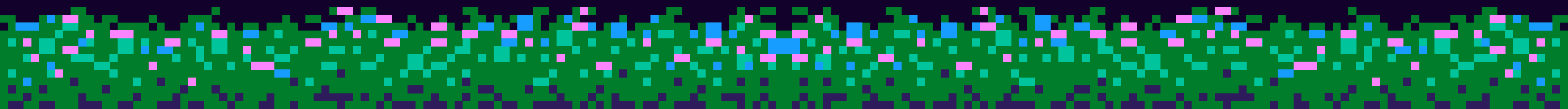
GITE

GITHUE

O QUE É GIT?

De longe, o sistema de controle de versão moderno mais usado no mundo hoje é o Git. O Git é um projeto de código aberto maduro e com manutenção ativa desenvolvido em 2005 por Linus Torvalds, o famoso criador do kernel do sistema operacional Linux. Um número impressionante de projetos de software depende do Git para controle de versão, incluindo projetos comerciais e de código-fonte aberto. Os desenvolvedores que trabalharam com o Git estão bem representados no pool de talentos de desenvolvimento de software disponíveis e funcionam bem em uma ampla variedade de sistemas operacionais e IDEs (Ambientes de Desenvolvimento Integrado).

Tendo uma arquitetura distribuída, o Git é um exemplo de DVCS (portanto, Sistema de Controle de Versão Distribuído). Em vez de ter apenas um único local para o histórico completo da versão do software, como é comum em sistemas de controle de versão outrora populares como CVS ou Subversion (também conhecido como SVN), no Git, a cópia de trabalho de todo desenvolvedor do código também é um repositório que pode conter o histórico completo de todas as alterações.



CONTROLE DE VERSÃO



COMO FUNCIONA?

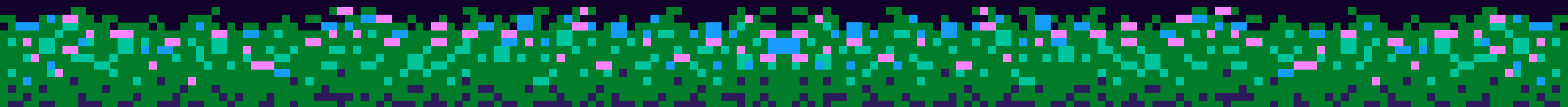
Todos os nossos arquivos, assim como seus históricos, ficam em um repositório e existem vários sistemas que gerenciam repositórios assim, como CVS (Sistema de Versões Concorrentes) e SVN (Subversion do Apache).

O Git é uma alternativa com um funcionamento mais interessante ainda: ele é distribuído e todo mundo tem uma cópia inteira do repositório, não apenas o "servidor principal".

E uma das vantagens disso é que cada pessoa pode desenvolver offline, realizando seus commits e outras operações sem depender de uma conexão constante com o servidor principal.

Mas...o que é a ferramenta Git exatamente?

O Git é um sistema de controle de versão distribuído e amplamente adotado. O Git nasceu e foi tomando espaço dos outros sistemas de controle.



COMO BAIXAR E INSTALAR?

Windows:

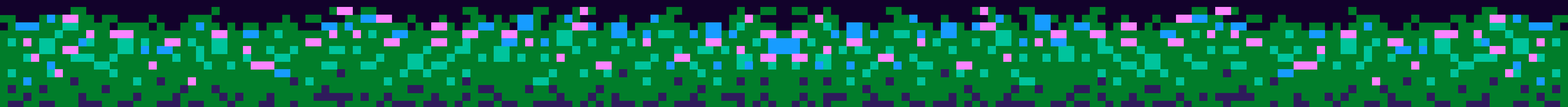
1. Acesse o site oficial do Git em "Todos os nossos arquivos, assim como seus históricos, ficam em um repositório e existem vários sistemas que gerenciam repositórios assim, como CVS (Sistema de Versões Concorrentes) e SVN (Subversion do Apache).
2. O Git é uma alternativa com um funcionamento mais interessante ainda: ele é distribuído e todo mundo tem uma cópia inteira do repositório, não apenas o "servidor principal".
3. É uma das vantagens disso é que cada pessoa pode desenvolver offline, realizando seus commits e outras operações sem depender de uma conexão constante com o servidor principal.
4. Mas...o que é a ferramenta Git exatamente?
5. O Git é um sistema de controle de versão distribuído e amplamente adotado. O Git nasceu e foi tomando espaço dos outros sistemas de controle.
6. "
7. Clique no link para download do Git para Windows.
8. Após o download, execute o instalador.
9. Siga as instruções do instalador, aceitando as configurações padrão, se não for um usuário avançado.
10. Conclua a instalação.

Linux:

1. No Linux, você pode instalar o Git usando o gerenciador de pacotes da sua distribuição. Por exemplo, no Ubuntu, use o comando `sudo apt-get install git`.
2. Se estiver usando outra distribuição, substitua o comando de acordo.

macOS:

1. No macOS, o Git pode ser instalado de várias maneiras, incluindo o uso do Xcode Command Line Tools, que geralmente já está instalado no sistema.
2. Abra o Terminal e digite `git --version` para verificar se o Git está disponível. Se não estiver, o sistema solicitará a instalação.
3. Siga as instruções para instalar o Git. Com esses passos simples, você pode instalar o Git no seu sistema operacional e começar a usar essa poderosa ferramenta de controle de versão.



VAMOS CONFIGURAR O GIT

1. Configure seu nome de usuário e e-mail:

- O Git registra quem fez cada alteração no código. Portanto, é importante configurar seu nome de usuário e e-mail. Use os comandos, no terminal:



```
git config --global user.name "Seu Nome"
git config --global user.email "seu@email.com"
```

2. Crie um Repositório Git:

- Para começar a rastrear seu código, crie um repositório Git em seu projeto. Navegue até a pasta do seu projeto e execute:



```
git init
```

3. Adicione Arquivos ao Controle de Versão:

- Use o comando git add para adicionar arquivos ao "staging area", que é onde você prepara os arquivos para serem "commitados" ou salvos.



```
git add nome-do-arquivo
```

4. Faça um Commit:

- Um commit é um snapshot de suas alterações. Use o comando git commit para criar um commit com uma mensagem descritiva do que foi alterado no projeto.



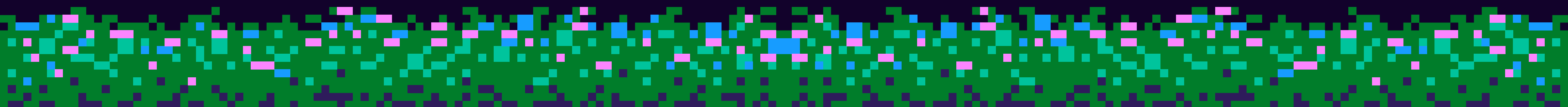
```
git commit -m "Sua mensagem de commit aqui"
```

5. Visualize o Histórico de Commits:

- Use git log para ver o histórico de commits no repositório



```
git log
```



CONCEITOS FUNDAMENTAIS DO GIT

Repositórios, commits e árvores (Trees)

Um repositório é como uma pasta ou diretório que contém todos os arquivos e o histórico de um projeto.

Já o termo commit pode ter como tradução literal “compromisso”, que seria uma ação em que você faz uma alteração no projeto, se compromete e salva suas alterações no histórico do projeto.

Ou seja, cada commit é uma entrada no histórico que contém informações sobre as alterações feitas.

Árvores, por último, representam a estrutura do diretório e arquivos em um commit específico, que tem como função registrar a organização do projeto ao longo do histórico de desenvolvimento.

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

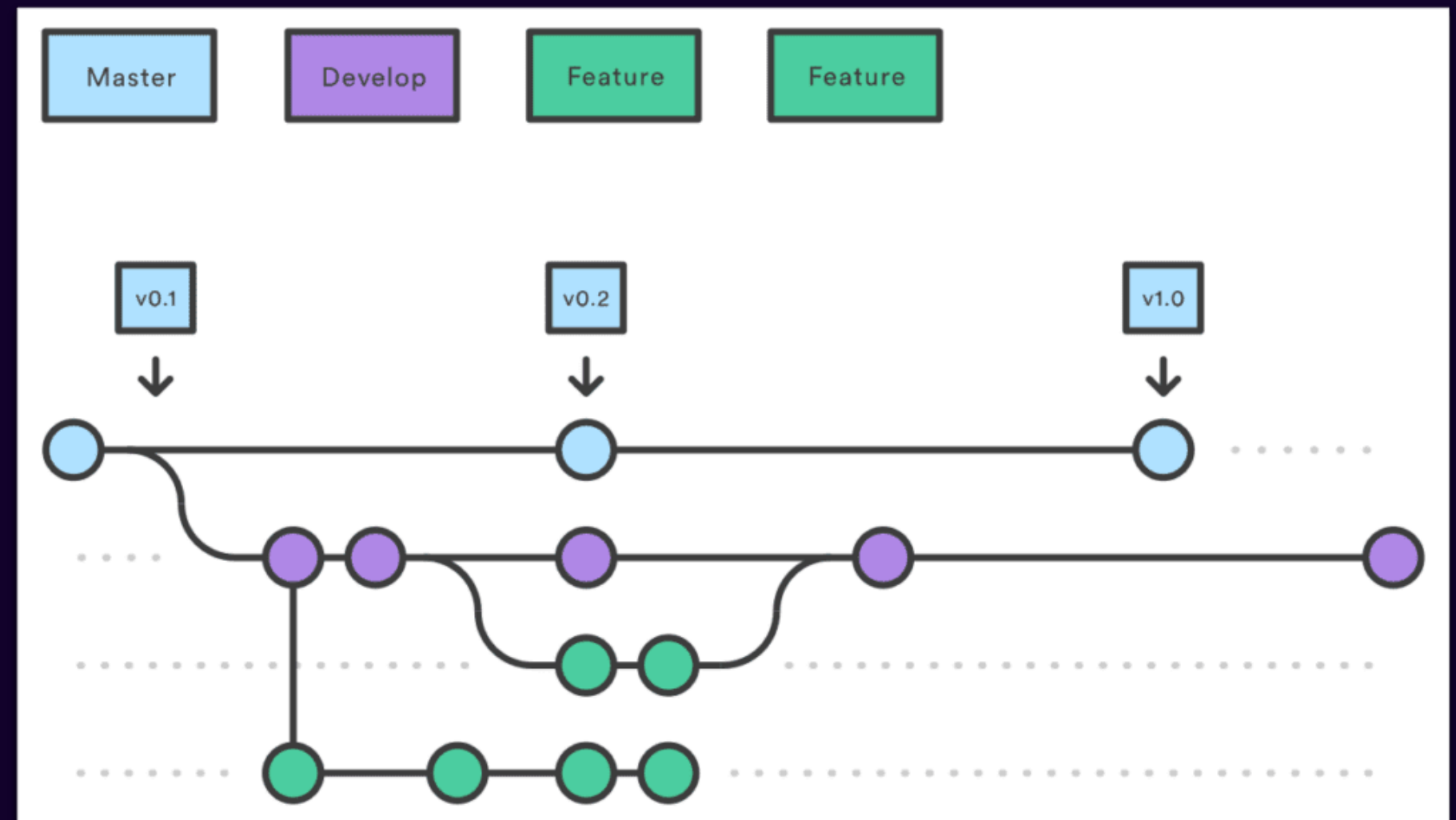
CONCEITOS FUNDAMENTAIS DO GIT

Ramificações (Branches) e fusões (Merges)

As “ramificações” ou branches permitem que você crie linhas separadas de desenvolvimento para trabalhar em recursos ou correções sem afetar a linha principal do projeto.

Cada branch é uma ramificação independente do código-fonte, possibilitando que você isole e desenvolva novas funcionalidades, refatore o código ou faça correções e testes em paralelo, sem interferir no código existente na branch principal, que geralmente é nomeada como "main".

Em um projeto com branches diferentes, a fusão, ou merge, permite combinar as alterações dessas branches de volta à linha principal, quando as alterações estão prontas.



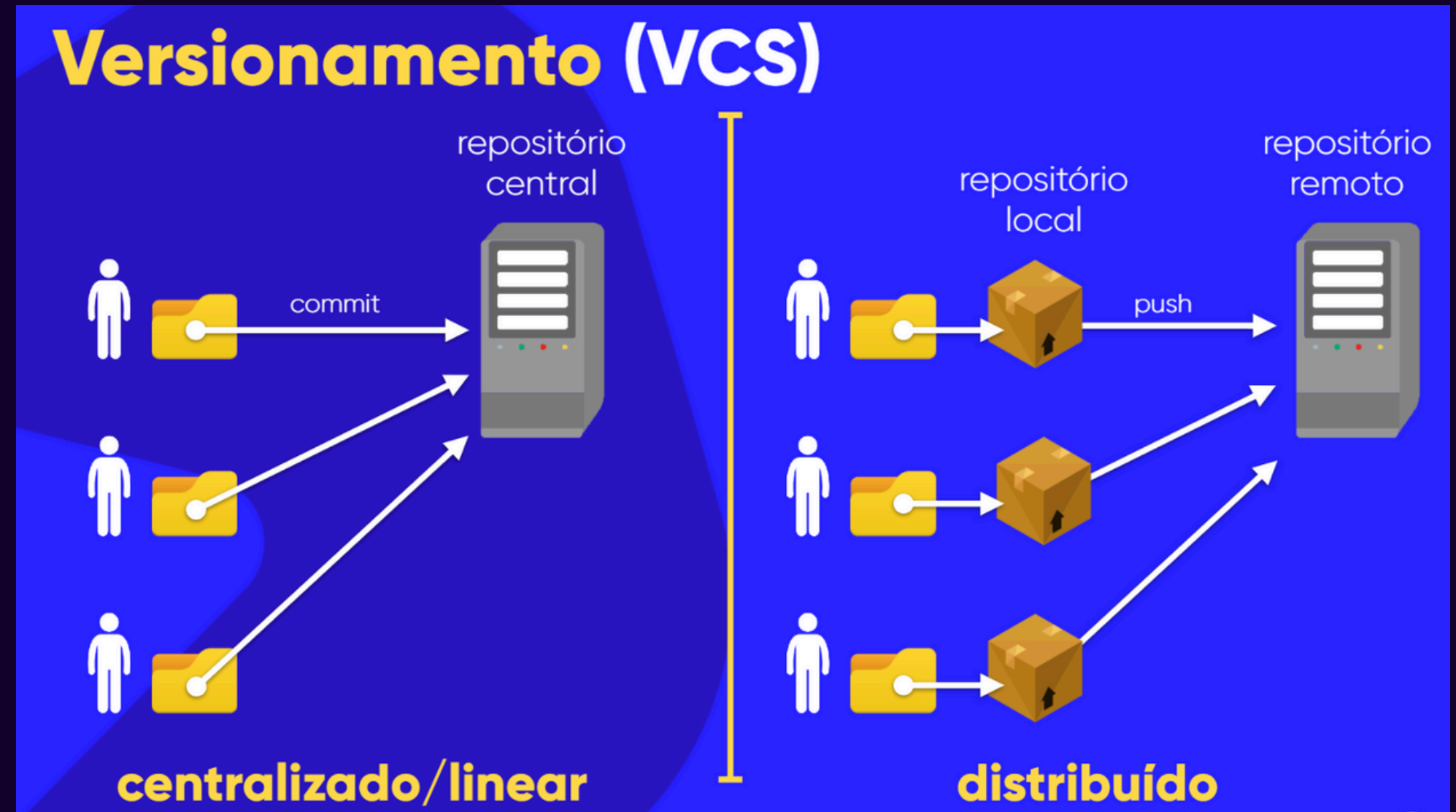
CONCEITOS FUNDAMENTAIS DO GIT

Controle de versão distribuído

Existem dois tipos de sistemas de controle de versão. Em um deles, pode haver um único servidor central que armazena o projeto com seu histórico, com o qual as pessoas desenvolvedoras precisam interagir. Isso é característico de um sistema de Controle de Versão Centralizado.

No outro tipo, cada pessoa desenvolvedora pode manter uma cópia do projeto em sua máquina local, o que é conhecido como Controle de Versão Distribuído, que é o caso do Git.

Com o Git, cada pessoa desenvolvedora tem uma versão completa do histórico do projeto. Isso proporciona independência e permite o desenvolvimento em paralelo.



GIT CENTRALIZADO VS DISTRIBUÍDO

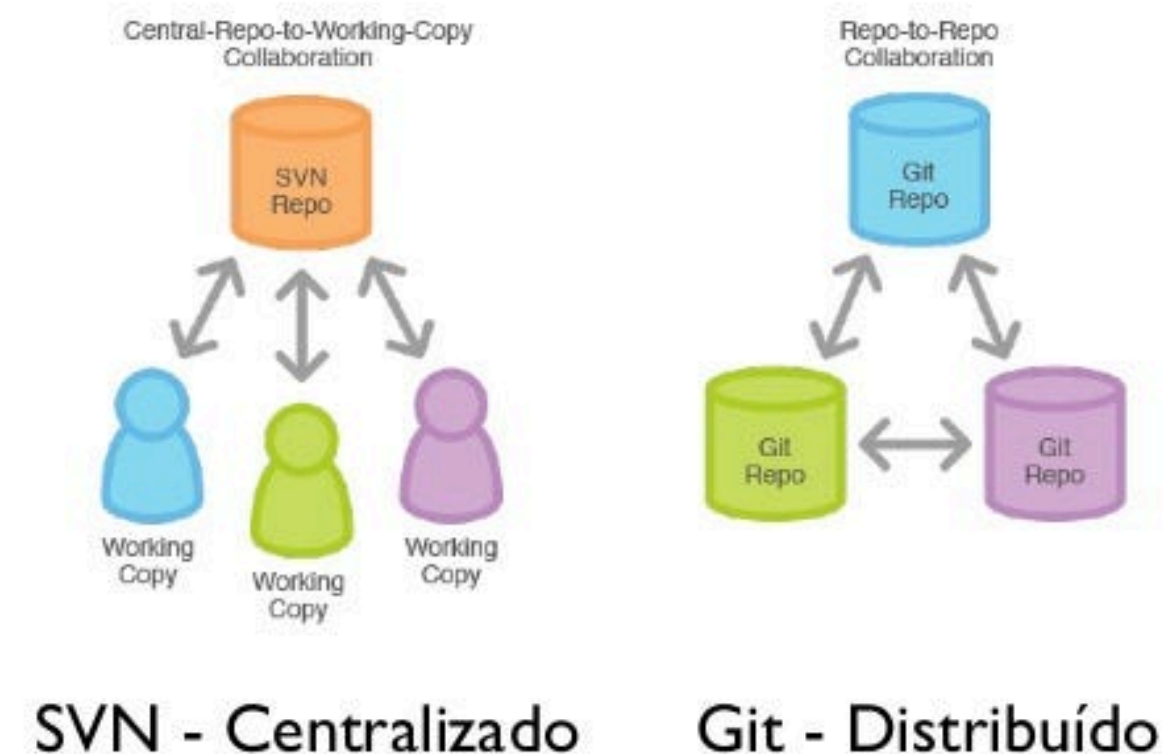
Controle de versão distribuído

Existem dois tipos de sistemas de controle de versão. Em um deles, pode haver um único servidor central que armazena o projeto com seu histórico, com o qual as pessoas desenvolvedoras precisam interagir. Isso é característico de um sistema de Controle de Versão Centralizado.

No outro tipo, cada pessoa desenvolvedora pode manter uma cópia do projeto em sua máquina local, o que é conhecido como Controle de Versão Distribuído, que é o caso do Git.

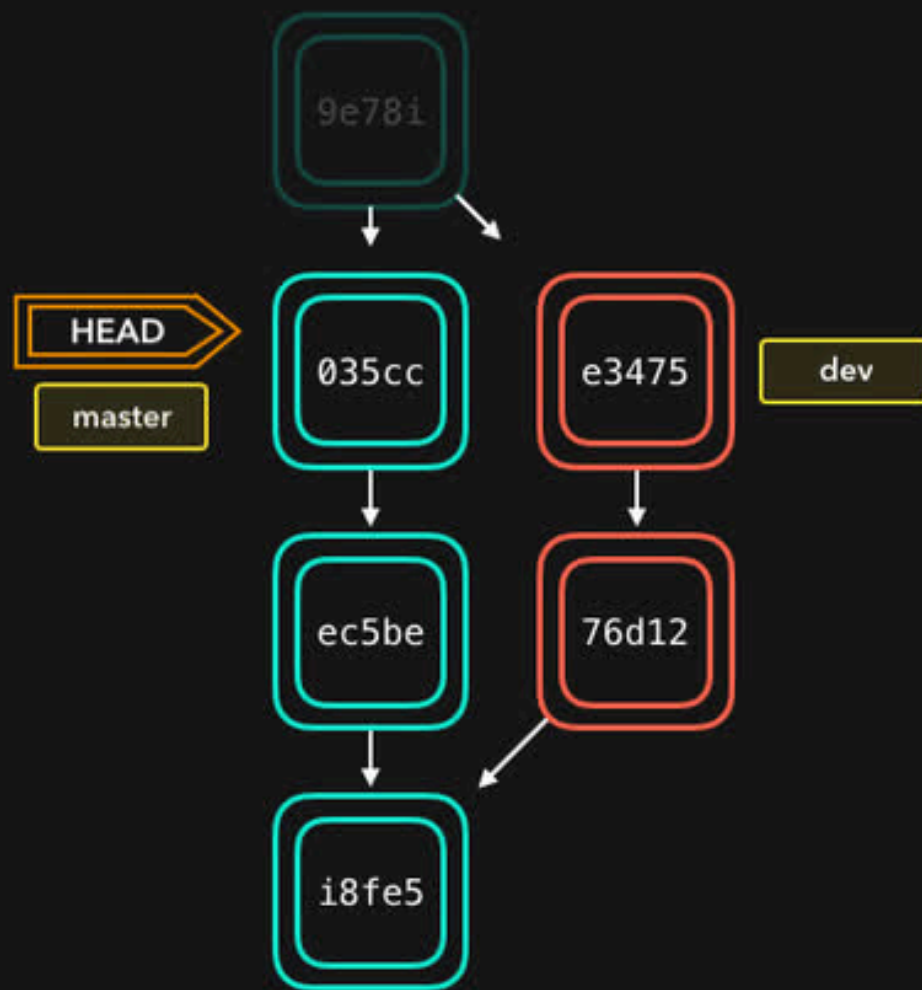
Com o Git, cada pessoa desenvolvedora tem uma versão completa do histórico do projeto. Isso proporciona independência e permite o desenvolvimento em paralelo.

GIT x SVN



COMMANDS

00 GIT



```
bash
master$ git merge dev
```

Git | Merging (no-fast-forward)

Default behavior when current branch contains commits that the merging branch doesn't have

Creates a new commit which merges two branches together without modifying existing branches

1) Git init

- É utilizado para inicializar um repositório Git dentro de um diretório do sistema. Após sua utilização, a ferramenta passa a monitorar o estado dos arquivos no projeto.

2) Git clone

- É utilizado para criar uma cópia de um repositório remoto em um diretório da máquina. Este repositório poder ser criado a partir de um repositório armazenado localmente, através do caminho absoluto ou relativo, ou pode ser remoto, através do URI na rede. A partir de um repositório clonado, é possível acompanhar o estado de um projeto e suas modificações, além de contribuir com o projeto, a partir do envio das suas modificações ao repositório central.

3) Git status

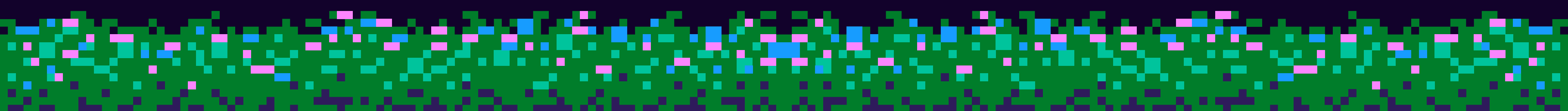
- É utilizado para verificar o status de um repositório git, bem como o estado do repositório central. O comando mostra informações sobre se o projeto local está sincronizado com o central, quais arquivos estão sendo monitorados pelo Git e em qual branch você está no projeto.

4) Git add

- É utilizado para adicionar arquivos ao pacote de alterações a serem feitas. É possível adicionar um único arquivo, múltiplos arquivos de uma vez, como `git add <-arquivo1-> <-arquivo2-> ...`, ou até mesmo um diretório, a partir de seu caminho. Uma vez que um arquivo é adicionado ao pacote de alterações com o comando `add`, ele está pronto para entrar no próximo commit.

5) Git commit

- É utilizado para criar uma nova versão do projeto a partir de um pacote de alterações. O commit pega o pacote de modificações adicionado através do comando `git add`, fecha essas alterações num pacote e o identifica através de um Hashcode. Além disso, para cada commit é necessário escrever uma mensagem para identificá-lo, com uma mensagem clara de quais alterações foram feitas neste commit.



6) Git log

- É utilizado para ver o histórico de alterações do projeto, onde aparecerão todos os commits feitos, com suas respectivas mensagens e códigos identificadores. O comando é muito útil quando precisamos rastrear o andamento de um projeto e verificar em qual ponto cada funcionalidade foi implementada. Além disso, o comando conta com várias opções para mostrar o histórico de forma resumida, gráfica e até mesmo mostrando a diferença entre os commits, que podem ser vistas na documentação oficial do comando.

7) Git branch

- É utilizado para criar novos ramos de desenvolvimento, bem como visualizar quais são os ramos existentes. Para criar um novo ramo, basta utilizar o comando `git branch` seguido do nome do novo ramo, e para visualizar quais os ramos existentes a utilização do comando é bem similar: basta não informar um nome para a nova branch, e serão listadas todas as já criadas.

8) Git checkout

- É utilizado para navegar entre as versões do projeto, bem como entre as diferentes ramificações criadas. Para navegar entre as versões, basta usar o comando:
 - `git checkout <- Hashcode do commit ->`
- E todo o estado do projeto se modificará ao estado no qual o commit foi feito. Similarmente, para navegar entre as ramificações podemos usar o comando:
 - `git checkout <- nome da branch ->`
- E a branch será alterada. O comando também permite criar uma branch e imediatamente mudar para ela, através do comando, que vai criar a ramificação e navegar até ela:
 - `git checkout -b <- nome da branch ->`

9) Git diff

É utilizado para visualizar modificações feitas entre commits, sejam eles entre um commit arbitrário e o estado atual do projeto, dois commits arbitrários, ou até mesmo todas alterações entre dois commits distintos.

Para visualizar as alterações entre um commit distinto e o atual, basta usar o comando:

```
git diff <- Hashcode do commit anterior ->
```

E serão listadas todas as diferenças no projeto entre os dois commits.

Para conhecer mais sobre o comando git diff e seus casos de uso, além de outros comandos e utilitários do Git, confira a [documentação do Git](#) e o [curso de Git e Github](#) da Alura.

10) Git config

- O comando git config é usado para configurar e personalizar o ambiente Git no seu sistema. Ele permite que você defina informações como seu nome de usuário, endereço de e-mail, editor padrão e muitas outras configurações que definem como o Git interage com seus repositórios. A estrutura básica do comando é:
 - git config <opções> chave valor
- Isso é útil para garantir que todos os commits que você fizer em qualquer repositório Git no seu sistema tenham o seu nome associado a eles. Além disso, o comando git config pode ser usado para personalizar muitos outros aspectos do seu ambiente Git, tornando-o mais adaptado às suas preferências e necessidades.

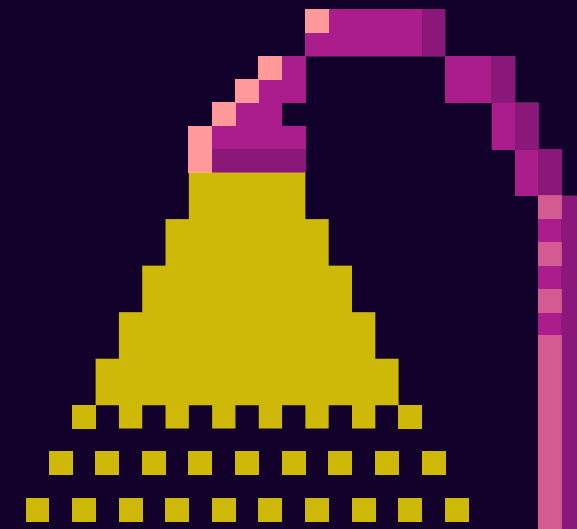
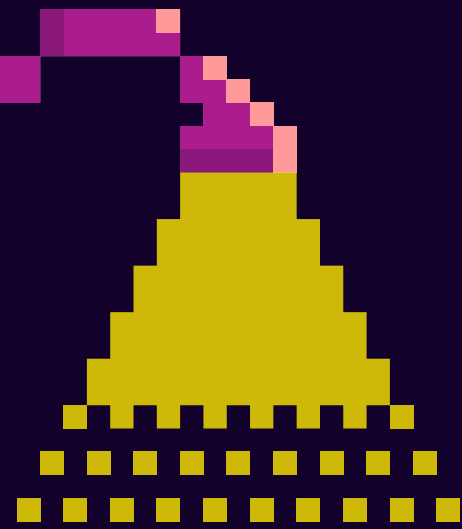
COMO O GIT ARMAZENA AS MUDANÇAS NO REPOSITÓRIO?

Quando você inicia um repositório Git com o comando `git init`, uma pasta oculta chamada **.git** é criada na raiz do projeto.

Essa pasta é o cérebro por trás do Git, onde todas as informações sobre as versões, histórico de commits e configurações são armazenadas.

A pasta `.git` contém três componentes principais:

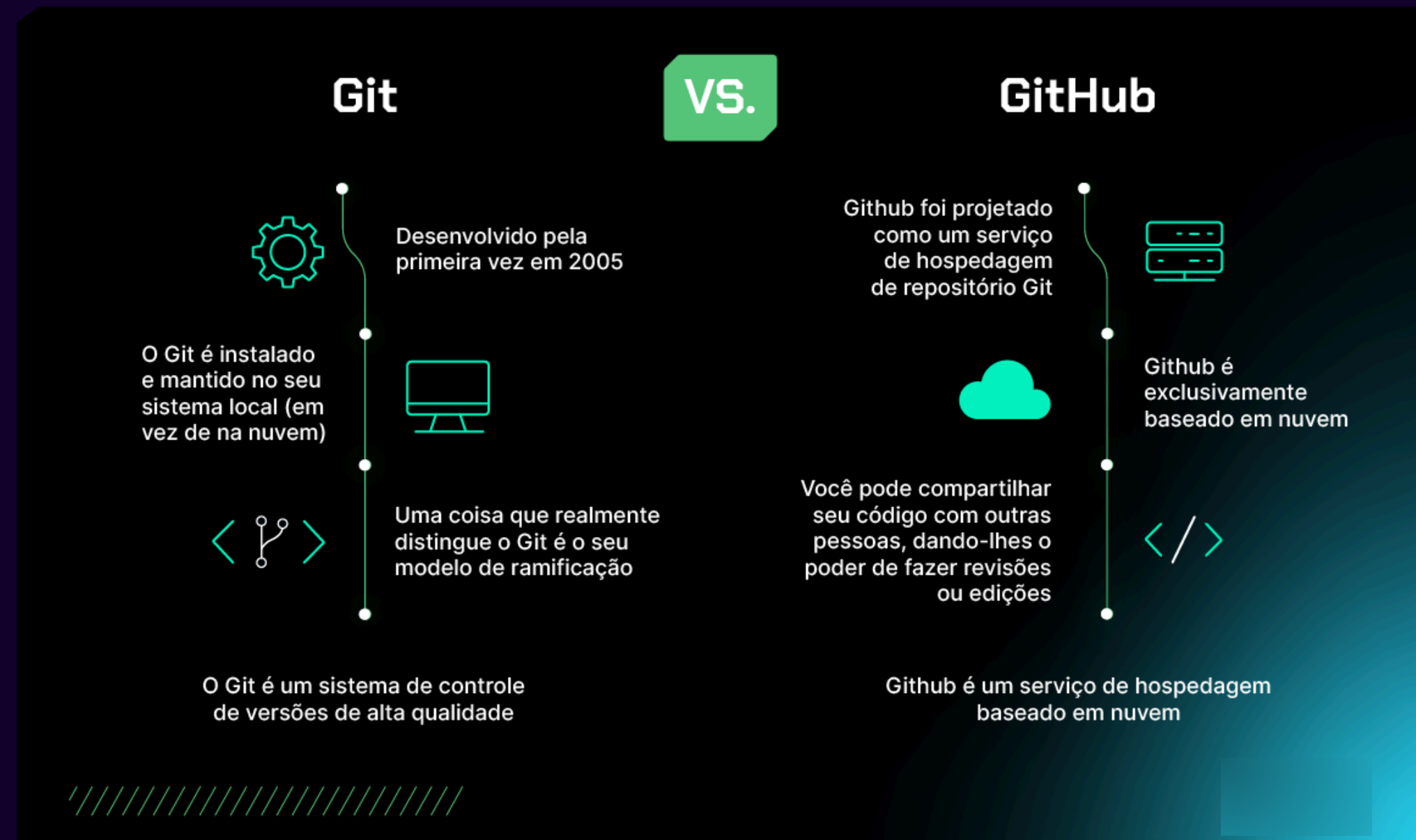
1. Diretório de objetos.
2. Diretório de referências.
3. Diretório de configuração.



O QUE É
GITHUE?



QUAIS AS DIFERENÇAS ENTRE ELES?





COMO CRIAR UMA CONTA?

Passo 1: Acesse o Site

Abra o seu navegador da web e acesse o site do GitHub em "<https://github.com>".

Passo 2: Iniciar a Criação da Conta

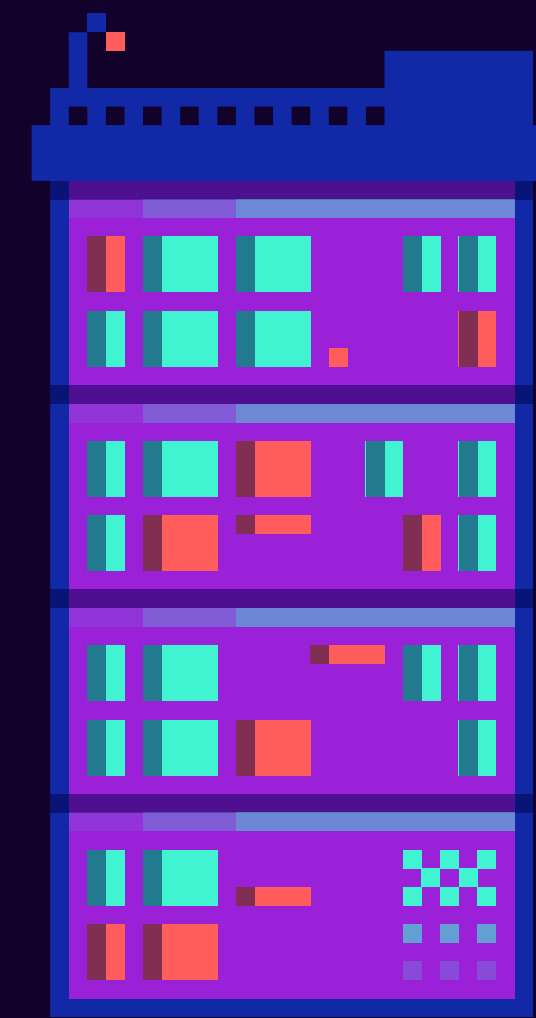
Na página inicial do GitHub, você encontrará no canto superior direito um botão "Sign up" (Inscrever-se). Clique nele para iniciar o processo de criação da conta.

Passo 3: Preencha suas Informações

Você será direcionado para uma página em que deve preencher suas informações pessoais, incluindo seu nome de usuário desejado, endereço de email e senha.

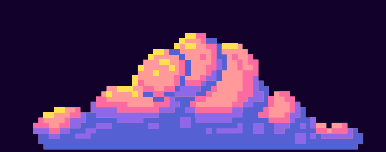
Passo 4: Verificação de Captcha

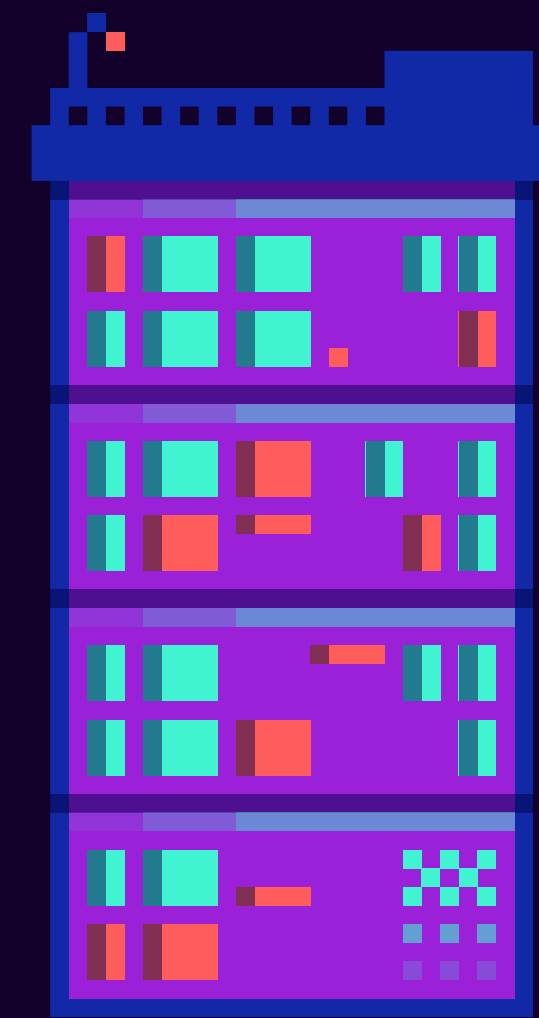
Para garantir que você não é um robô, o GitHub pode solicitar que você complete uma verificação de Captcha. Siga as instruções para provar que você é um usuário legítimo.



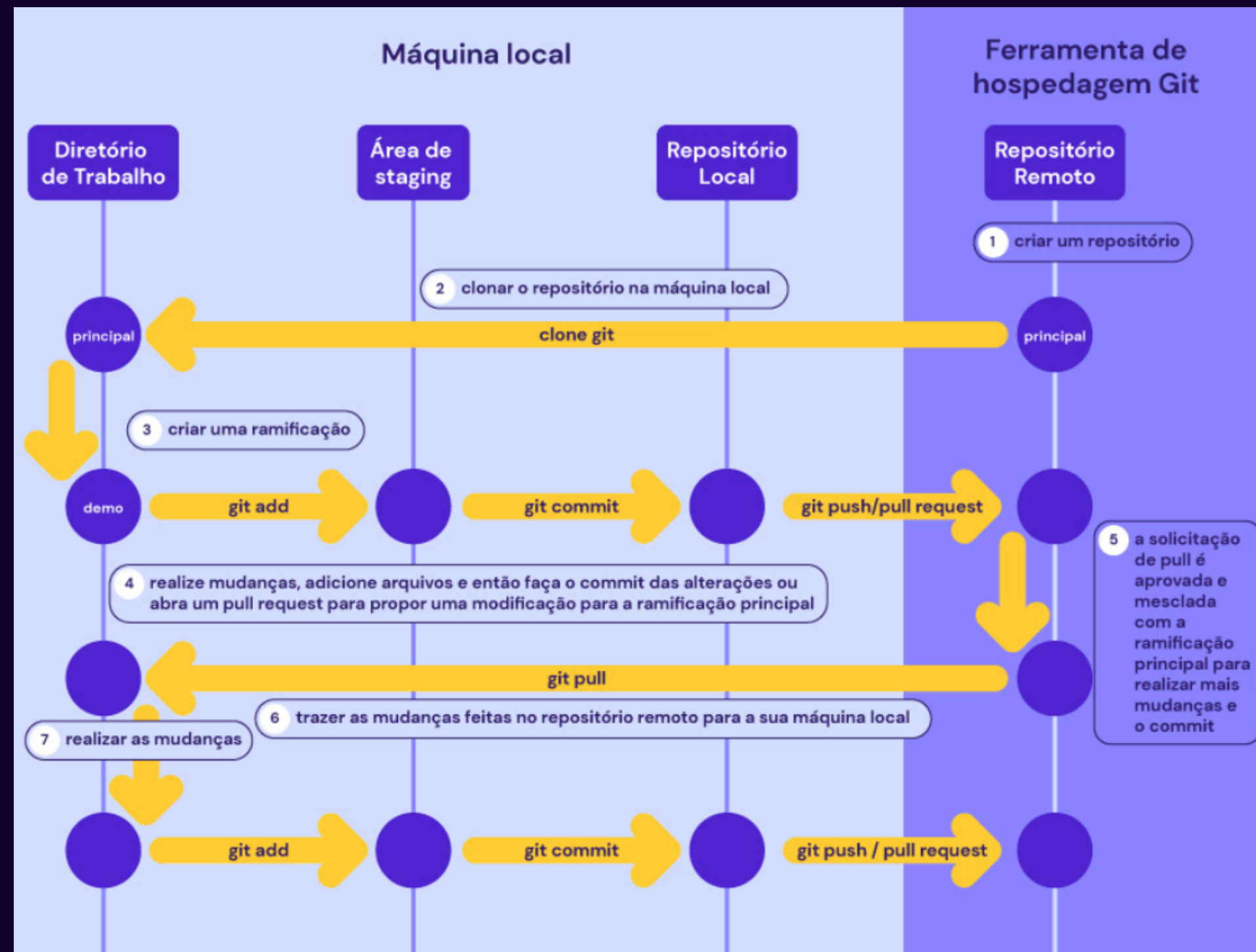


COMO CRIAR UM REPOSITÓRIO?

1. **Acesse sua Conta:** Certifique-se de estar logado na sua conta do GitHub. Se você não tiver uma conta, siga as etapas para criar uma, conforme explicado anteriormente.
 2. **Página Inicial:** Na página inicial do GitHub, clique no botão "New" (Novo) localizado no canto superior direito.
 3. **Nome e Descrição:** Preencha o nome do seu repositório e uma breve descrição. Escolha se deseja que o repositório seja público (visível para todos) ou privado (acessível apenas por convite).
 4. **Opções de Inicialização:** Você pode optar por inicializar o repositório com um arquivo README, que é uma boa prática para fornecer informações sobre o projeto. Além disso, você pode escolher uma licença para o seu código, se desejar.
 5. **.gitignore:** Você pode especificar tipos de arquivos que o Git deve ignorar ao rastrear alterações. Por exemplo, você pode selecionar uma linguagem de programação específica para gerar um arquivo .gitignore correspondente.
 6. **Escolha um Template (Opcional):** Se o seu projeto se encaixa em um dos modelos de projeto disponíveis, você pode escolher um para iniciar com estrutura pré-definida.
 7. **Create Repository:** Após preencher todas as informações necessárias, clique no botão "Create repository" (Criar repositório) para criar o seu repositório.
- 



REPOSITÓRIOS LOCAIS VS REPOSITÓRIOS REMOTOS



O QUE É README?

O **README** é um arquivo com extensão **.md**, ou seja, ele é escrito em Markdown que é uma linguagem de marcação utilizada para converter o texto em um HTML válido. Caso queira saber mais sobre, temos esse [artigo](#) que explica muito bem como funciona e como escrever anotações com essa linguagem.

01

- Descrição do seu projeto;

02

- Funcionalidades

03

- Como os usuários podem utilizá-lo;

04

- Onde os usuários podem encontrar ajuda sobre seu projeto;

05

- Autores do projeto.

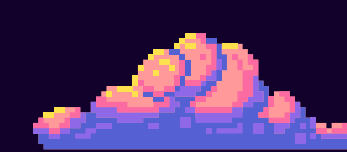


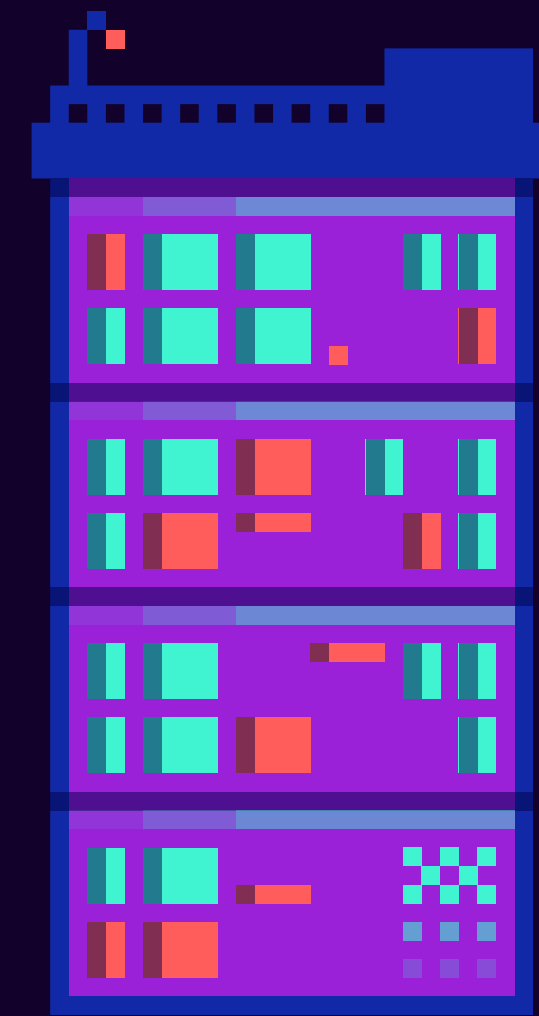


COMO BAIXAR NOVOS COMMITS DO REPOSITÓRIO REMOTO?



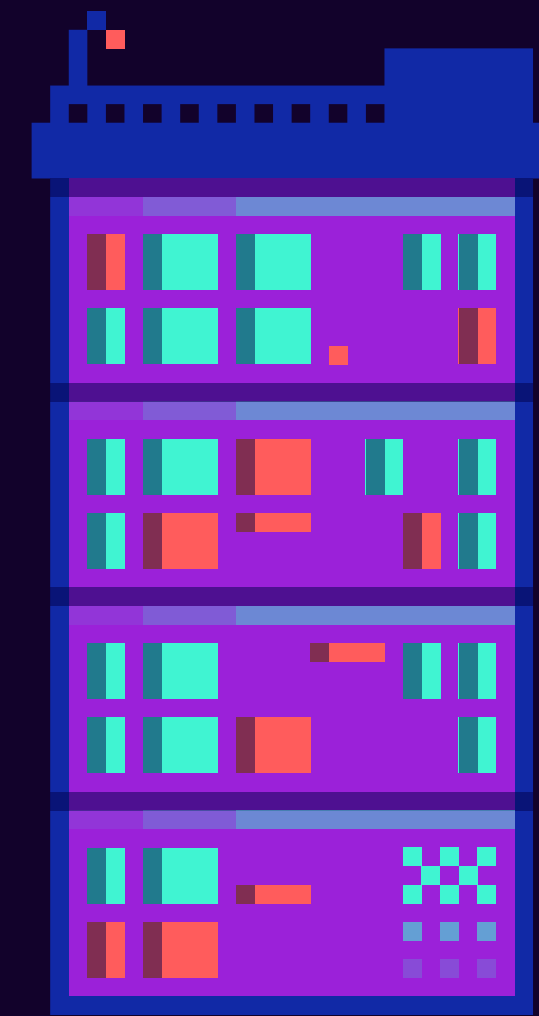
Trabalhando com o GitHub, em uma equipe, diversas versões podem ter atualizações diferentes entre si a todo momento, nesse caso, manter-se atualizado com as alterações feitas por outras pessoas colaboradoras é crucial:



1. Abra o Terminal ou Prompt de Comando: Semelhante aos outros passos, façamos os comandos no terminal (Linux, macOS) ou GitBash (Windows).
 2. Navegue até o Diretório do Repositório Local: Use o comando **cd** para navegar até o diretório do seu repositório local.
 - a. **cd <caminho/do/seu/repositorio>**
 3. Atualize o Repositório Local com os Novos Commits: Utilize o comando **git pull** para buscar os novos commits do repositório remoto e atualizar sua branch local.
 4. Resolva Conflitos (Se Aplicável): Se ocorrerem conflitos durante o processo de atualização, o Git notificará você. Nesse caso, será necessário resolver os conflitos manualmente antes de continuar. As interfaces de IDEs (Ambientes de desenvolvimento integrado) ou o próprio GitHub, oferecem uma visualização otimizada de onde estão os conflitos para que você possa resolver.
 5. Verifique as Alterações Locais: Após o **git pull**, você pode verificar as alterações locais usando **git log** para visualizar os novos commits no histórico do seu repositório.
- 



COMO SEPARAR O DESENVOLVIMENTO DE DIFERENTES FUNCIONALIDADES?

1. **Branches (Ramificações):** As branches podem ser utilizadas para isolar o desenvolvimento de diferentes funcionalidades. Cada branch representa uma linha independente de desenvolvimento. Por exemplo, você pode ter uma branch para desenvolver uma nova funcionalidade, outra para corrigir um bug e assim por diante.
 - a. Comando para criar uma nova branch para uma nova funcionalidade: **git branch nova-funcionalidade**
 - b. Mudar para a nova branch: **git checkout nova-funcionalidade**
 2. **Git Flow:** Considere adotar o modelo Git Flow, uma abordagem popular para organizar branches em um projeto. Ele define branches específicas para desenvolvimento, releases e features, o que facilita o gerenciamento de diferentes aspectos do ciclo de vida do software.
 3. **Feature Branches (Branches de Funcionalidades):** É comum a utilização de branches específicas para cada funcionalidade que está sendo desenvolvida. Isso mantém as alterações relacionadas a uma funcionalidade isoladas de outras partes do código. Ex.: **git checkout -b feature/nova-funcionalidade**
 4. **Git Merge:** Após completar o desenvolvimento em uma branch de funcionalidade, você pode mesclar as alterações de volta para a branch principal. Isso integra a nova funcionalidade ao código principal.
 - a. Mudar para a branch principal: **git checkout main**
 - b. Em seguida, mescle as alterações da feature de volta para a branch principal **git merge feature/nova-funcionalidade**
 5. **Pull Requests (Solicitações de Pull):** Em ambientes colaborativos, é utilizado **pull requests** para revisar e discutir as alterações antes de mesclá-las de volta à **branch** principal. Isso adiciona uma camada extra de controle de qualidade e colaboração ao processo.
- 

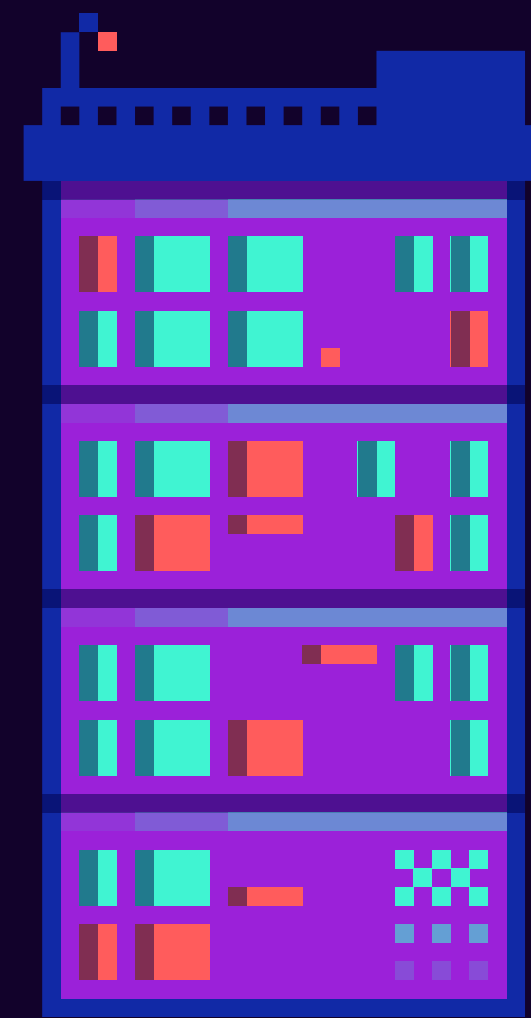
BRANCHES E MERGES

As branches (ramificações) no Git são uma ferramenta poderosa para organizar o desenvolvimento, permitindo que diferentes linhas de código evoluam independentemente.

Mas quando terminamos de desenvolver as funcionalidades para uma branch?

Como unimos as alterações com a branch principal? Para isso, existe o merge, com ele, quando a funcionalidade estiver pronta, ela será incorporada à branch principal, exemplo:

- Primeiro é preciso, ir para a branch principal
 - `git checkout main`
- Mesclar as alterações da funcionalidade de volta para a branch principal, com:
 - `git merge feature/nova-funcionalidade`



FORK/PULL REQUEST

O processo de fork e pull request é fundamental para contribuições em projetos de código aberto, permitindo que outras pessoas sugiram alterações sem afetar diretamente o repositório original.

- Fork: Em um projeto de código aberto, clique no botão "Fork" no canto superior direito da página.
- Isso cria uma cópia do repositório original na sua conta.
- Clonando o Fork para o Repositório Local:

Feito isso, para fazer alterações no código, é preciso ter esse código em sua máquina de forma local, que é possível da seguinte forma:

- **git clone https://github.com/seu-usuario/nome-do-fork.git**
- **cd <caminho/até/pasta-com-o-nome-do-fork>**

Em seguida, é uma boa prática você criar uma ramificação para armazenar suas alterações, sem prejudicar o andamento do projeto principal, dessa forma:

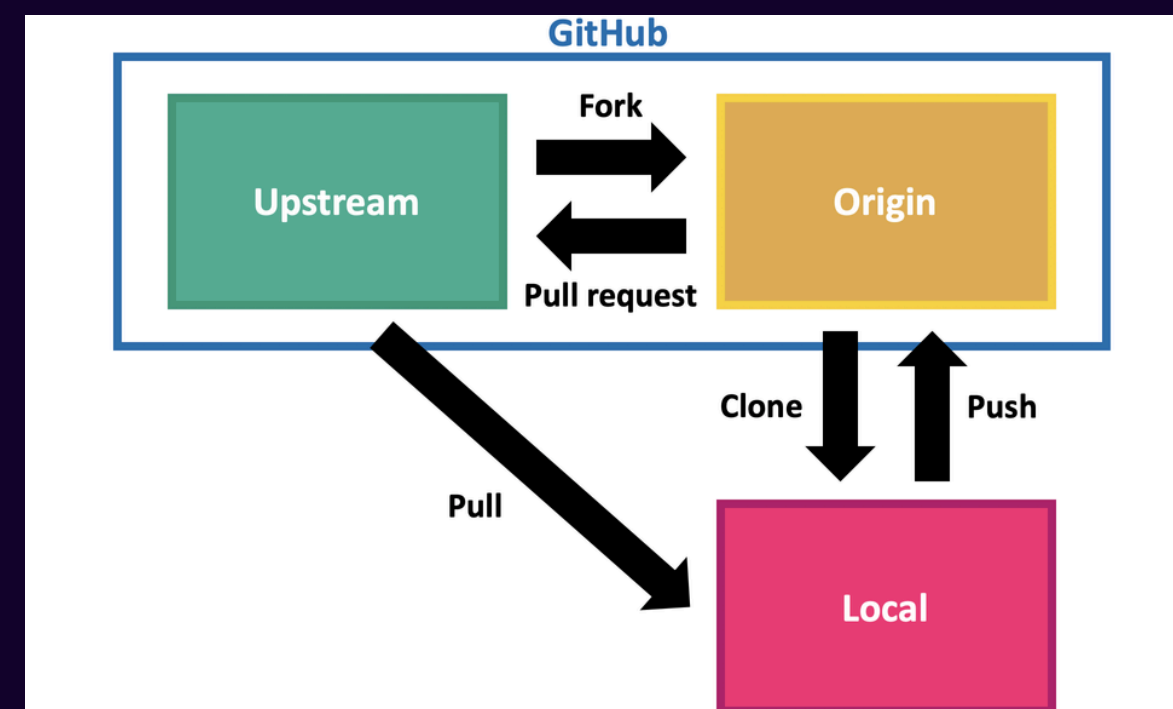
- **git checkout -b feature/minha-contribuicao**

Realize as alterações desejadas, faça commits e, se necessário, crie novas branches para diferentes funcionalidades ou correções.

Feito isso, para enviar as alterações para seu repositório com a versão do fork, faça:

- **git push origin feature/minha-contribuicao**

Por fim, para concretizar suas alterações no projeto, no GitHub, vá até o seu fork e clique em "New Pull Request". Escolha a branch com suas alterações e sugira a incorporação ao repositório original.



GIT COPILLOT



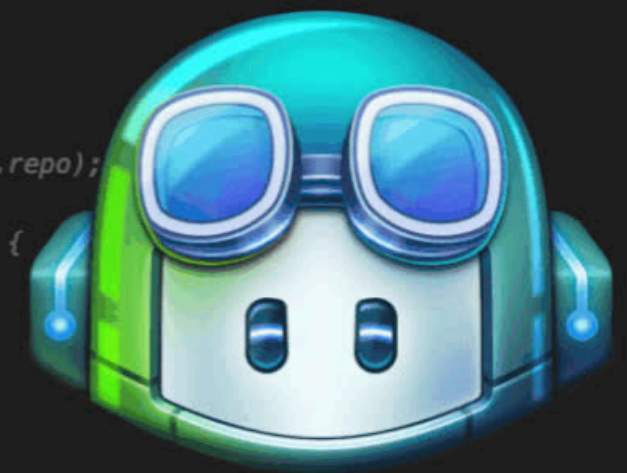
```
JS test.js 1 X
Users > basimhennawi > Desktop > JS test.js > github
1 function githubCopilot(config) {
  var github = new Github({
    token: config.token,
    auth: 'oauth'
  });

  var repo = github.getRepo(config.user, config.repo);

  var commits = repo.listCommits(config.branch, {
    sha: config.sha
  });

  commits.on('data', function(commit) {
    console.log(commit.sha);
  });

  commits.on('error', function(err) {
    console.log(err);
  });
}
```



MEUS CONTATOS:



27 99500-7495



<https://beacons.ai/prismatech>



producaoprismatech@gmail.com



Avenida Jerônimo Monteiro 145, Vitória





THANK
YOU