1. Curso de git

1.1. O que é?

Git é um sistema de controle de versão open-source, ou seja, gratuito. Ele é utilizado para a criação de um histórico de alterações em código-fonte de projetos de desenvolvimento de software. Foi desenvolvido por Linus Torvalds, o criador do sistema operacional Linux.

Através dele podemos desenvolver projetos na qual diversas pessoas podem contribuir simultaneamente no mesmo, editando e criando novos arquivos e permitindo que os mesmos possam existir sem o risco de suas alterações serem sobrescritas, além de saber quais foram as alterações realizadas, quem fez cada uma das alterações e baixar essas mudanças em nossa máquina. E caso seja necessário, revertê-las para uma versão anterior.

1.2. Como instalar?

- Windows: https://git-scm.com/download/windows
- MacOs: `$ brew install git`
- Debian/Ubuntu: `$ apt-get install git`
- Fedora: `$ dnf install git`
- Gentoo: `$ emerge --ask --verbose dev-vcs/git`
- Arch Linux: `$ pacman -S git`
- openSUSE: `$ zypper install git`

1.3. Principais Conceitos

- **Commit:** Realiza uma mudança no projeto; mais específicamente, armazena uma mudança no banco de dados de uma forma que possa ser incorporada em versões futuras.
- **Update:** Solicitar que as mudança dos demais commits sejam incorporadas em sua cópia local do projeto.
- **Branch (Ramo):** Uma cópia do projeto, porém isolada de uma maneira que as mudanças realizadas no branch não afetem o resto do projeto e vice-versa, *exceto quando as mudanças são delibaradamente mescladas*
- **Repositório:** Onde armazena-seo histórico do projeto.
- **Diretório de Trabalho:** É um diretório monitorado pelo controle de versão que contém os arquivos do projeto.
- **Configuração:** É o estado dos arquivos do projeto.
- **Revisão:** É uma configuração registrada no
- repositório.
- **Rastreabilidade:** É poder seguir o trajeto de uma solicitação de mudança desde que foi proposta até o momento que voi implementada.
- **Changeset:** Diferença entre duas configurações.
- **Versão:** Revisão usada em produção.
- **Merge:** Aplica os commits de uma branch para a branch atual, encontra um commit em comum entre as branches (base) e adiciona os commits que a branch atual não posui (caso não existam) em um commit de merge.
- **Rebase** É semelhante ao merge porém é diferente no modo em que os commits são aplicados. No rebase, os commits a frente da base são temporariamente removidos e os commits da outra branch são aplicados, por os commits da branch são adicionados.
- **Fetch:** Baixa as atualizações do remoto, mas não aplica elas.
- **Tags:** Uteis para definir versões do projeto, semelhante as branches porém não recebe mais commits, guardando um estado do repositório.

1.4. Serviços Fundamentais

- Registro da evolução do projeto;
- Controle de concorrência;
- Variações do projeto.

> É uma das ferramentas mais importantes do desenvolvimento de software.

Serve para:

1. Manter histórico do projeto;
2. Controlar a concorrência de edição;
3. Manter váriações do projeto.

1.5. Níveis de Configuração

- **Sistema:** É o mais abrangente vale para todos os usuários e repositórios, geralmente é usado em servidores de repositórios.
- **Global:** Vale para todos os repositórios do usuário, a configuração deste nível sobreescreve a configuração de sistemas.

- **Local:** É o mais específico e vale apenas para o repositório que está sendo usado, sobreescrevendo as configurações dos outros níveis. A configuração do nível local costuma ser gerada automaticamente durante a clonagem ou inicialização do repositório, geralmente contém o caminho original para o repositório que é usado na sincronização de repositórios e etc.

1.6. Arquivos de configuração (Linux)

- **Local:** `repositório/.git/config`
- **Global:** `$HOME/.gitconfig`
- **Sistema:** `/etc/gitconfig`

1.7. Workflow

- Editar
- Commitar
- Sincronizar com o repositório

1.8. Estados dos Arquivos

Estado	Stage
Não Monitorado	Untracked
Modificado	Modifield
Preparado	Staged
Consolidado	Commited

1.9. Comandos

1.9.1. Configuração Básica

- `git config --global user.name nome` Definir nome do usuário.
- `git config --global user.email email@email.com` Definir e-mail do usuário.
- `git config --global core.editor editor` Definir editor de texto.
- `git config --list` Listar todas as configurações
- `git config --list --global` Lista todas as configurações globais.
- `git config --global --edit` Abre o arquivo de configurações para edição.
- `git config --global credential.helper store` Salva as credênciais do git.

1.9.2. Comandos Básicos

- `git init` Inicializa um novo repositório.
- `git init --bare` indica que é um repositório "puro", ou seja, com tém apenas as alterações dos arquivos, e não uma cópia deles
- `git status` Mostra o estado dos arquivos dentro do repositório.
- `git log` Lista todos commits feitos no repositório.
- `git log --graph` Lista todos commits feitos no repositório junto com a reprentação dos branches.
- `git show commit` Mostra as informaçãoes do commit.

- `git add <arquivo>` Adiciona determinado arquivo para o estado de staged.
- `git add .` Adiciona todos os arquivos para o estado de staged.
- `git commit -m "mensagem"` Guarda uma versão para o repositório.
- `git commit --amend` Altera o último commit, tanto a mensagem de commit quanto a adição de arquivos.
- `git push` Envia os commits para um repositório remoto.
- `git diff` Apresenta as diferenças entre commits.
- `git diff HEAD~1` Mostra as diferenças da versão atual (HEAD) com a versão anterior.
- `git blame <arquivo>` Mostra as alterações feitas em um arquivo linha por linha. Mostra o autor e o commit onde foi feita aquela linha.

1.9.3. Repositórios Remotos

- `git remote` Lista o nome dos repostórios remotos configurados para o repositŕio local.
- `git remote -v` Lista o nome dos repostórios remotos configurados para o repositŕio local, e mostra os endereços de cada remoto.
- `git remote add <nome> <local>` Adiciona um repositório remoto. O local pode ser uma URL, o IP de uma máquina na rede ou até um diretório local.
- `git remote rename <nome atual> <novo nome>` Renomeia o repositório remoto.

1.9.4. Histórico e Conflitos

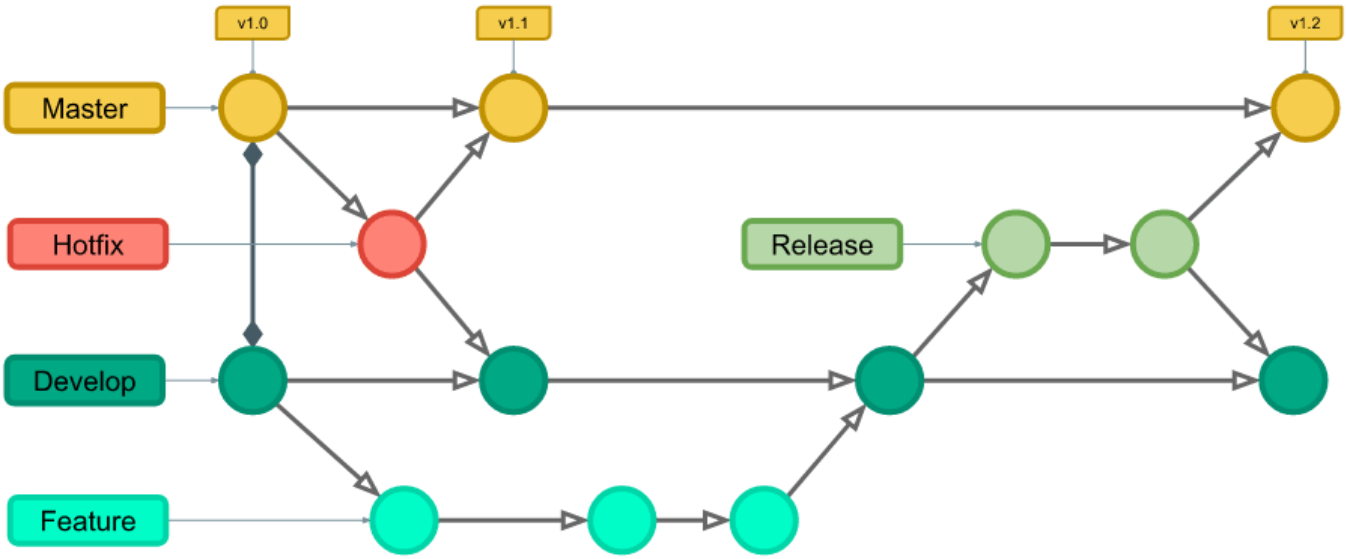
- `git clone https://github.com/usuario/repositorio.git` Baixa o repositório remoto, serve como uma maneira alternativa de inicializar um repositório e já vem com o remote configurado.
- `git pull` Baixa as alterações do repositório remoto que não se encontram no repositório local. *mantem o repositório sincronizado com os ultimos commits de uma branch.*
- `git checkout <commit> --arquivo` Permite ver o estado de um arquivo ou todo o diretório em um determinado commit.
- `git checkout --arquivo` Ignora as mudanças feitas no arquivo *que não estejam em staged.*
- `git checkout HEAD --arquivo` Desfaz todas as alterações até o ultimo commit *incluindo os arquivos em staged.*
- `git revert <commit>` Cria um novo commit desfazendo as alterações de um commit específico.
- `git reset` Reseta o repositório para um determinado commit (por padrão, usa-se --soft).
    - `git reset --soft` Ignora o commit, mas as modificações no arquivo continuarão e o mesmo se encontra no estado de staged.
    - `git reset --mixed` Ignora o commit, mas o arquivo estará no estado de modified
    - `git reset --hard` Ignora tudo no commit.
- `git bisect` Faz a busca entre determinados commits em que a alteração desejada foi alterada
    - `git bisect start` Inicia a busca
    - `git bisect bad <commit>` Estado em que o código possuí algum bug ou erro (Inicio da busca)
    - `git bisect good <commit>` Informa o estado desejado
    - `git bisect bad` Informa o estado não é desejado
    - `git bisect reset` retorna para a branch

1.9.5. Branching, Merge e Rebase

- `git branch` Lista todas as branches.

- `git branch <nome>` Cria uma nova branch.
- `git branch -d <nome>` Remove uma branch.
- `git checkout <branch>` Altera para uma branch.
- `git merge <branch>` Faz o merge entre as branches.
- `git rebase <branch>` Faz o rebase entre as branches.
- `git rebase --continue` Passa para o próximo commit (caso tenha mais conflitos)
- `git rebase --abort` Aborta todo o processo de rebase
- `git fetch` Faz o fetch no repositório.
- `git tag [nome tag]` Cria uma tag.
- `git push <remoto> <tag>` Envia a tag para o repositório remoto.
- `git fetch origin pull/<ID>/head:<branch>` faz o checkout de um PullRequest.
- `git stash` Guarda as alterações do *Working Directory*. Permite fazer o rebase, marge e trocar de branch sem a necessidade de fazer um commit.
- `git stash list` Lista os stashes.
- `git stash pop` Aplica o último stash.
- `git cherry-pick <commit>` Aplica as alterações de um commit na branch atual.

1.9.6. Dicas de branching (Gitflow)



- `master` Onde fica o código em produção
- `development` Onde todas as novas features são mergeadas
- `feature/<nome-da-feature>` Nessas branches, devem ser adicionadas as novas funcionalidades do sistema, que no fim, irão para `development`.
- `hotfix/<versão>` Onde será criado as correções de bugs que ocorrem na produção, por padrão essas branches partem da `master`.
- `release/<versão>` Onde será enviado o código de homologação.

1.9.7. .gitignore

- Configura os arquivos que devem ser ignorados.
- Contém arquivos, caminhos e patterns.
  - `.project, .jar, .zip` Ignora arquivos com uma determinada extenção.
  - `.[jw]ar` Usa uma expressão regular para ignorar arquivos de uma determinada extenção.

    - `dist/` Ignora diretórios.
    - `**/log/` Ignora qualquer diretório que tenha um subdiretório semenhante ao específicado.
    - `**/*.css` Ignora qualquer caminho que termina com um arquivo de uma determinada.

1.9.8. Dicas e Boas práticas

- Nunca faça o commit de um código que não funciona.
- Um commit pode ser realizando quando:
    - Um bug for solucionado
    - Uma feature foi adicionada
    - Após a realização de uma tarefa
    - No fim do dia
        - Todas esses cenários são válidos desde que o código esteja em funcionamento