Resumo Git

# O que é Git?

Git é um **Sistema de Controle de Versão** e de **Código Aberto**, criado por Linux Torvalds em 2005.

# Sistema de Controle de Versão:

Na área de Tecnologia da informação, o Controle de Versão é um sistema que armazena todo o histórico no desenvolver de um projeto de softwares, seja lá qual for a alteração no código, inclusão, remoção, etc. Resumido, sua função é gerenciar o versionamento de um documento, que ao longo de seu desenvolvimento sofrerá alterações ao longo do tempo.

É importante registrar que o Git, ao contrário do que muitas pessoas pensam, não precisa ser utilizado apenas para versionamento de códigos-fonte, mas também pode ser utilizado para qualquer outro tipo de arquivo, assim como um simples documento Word.

Ter o controle de versão de um documento é ter a possibilidade de responder perguntas como: “O que mudou?”, “Por que mudou?”, “Quando mudou?”, “Quem mudou?”, entre outras.

## Sistema de Controle de Versão Local

É considerado um dos métodos mais simples de versionamento. Basicamente você copia e coloca os arquivos em diferentes diretórios na Máquina Local, colocando sempre documentos com data, hora, o que mudou, o que foi adicionado, quem mudou, etc.



## Sistema de Controle de Versão Centralizado

O Sistema de Controle de Versão Local tinha um grande problema: Ele dificultava o trabalho em equipe, já que o versionamento estava na máquina local. Portanto, o Sistema de Controle de Versão Centralizado possibilitou através da arquitetura Client-Server – onde temos uma máquina central (Servidor) que contém todas as versões do arquivo e as máquinas (Clientes) que são capazes de Gravar e Resgatar verões do Servidor – que foi estável e extremamente utilizada até pouco tempo atrás. Seu declínio ocorreu devido a extrema dependência de uma máquina central, pois, se ela se tornasse inoperável ou fosse corrompida, os dados se tornariam inacessíveis para todos os Cliente.



## Sistema de Controle de Versão Distribuído

Após detectar os problemas do Sistema de controle de Versão Centralizado, foi construído o Distribuído, que se baseia na cópia de todas as versões para cada desenvolvedor, não dependendo mais de um Servidor que detém todas as versões. Isso possibilita que o desenvolvimento em equipes se tornem mais práticos e, de certa forma, simultâneos.



Além de controlar as versões dos documentos e programas, o Git possibilita uma alta segurança para o armazenamento do código-fonte, utilizando criptografia Hash do tipo SHA1. Isso protege o código e seu histórico, além de ser altamente rastreável.

# Conceitos:

## Repositório

O repositório é onde estarão os arquivos versionados, no servidor Git e no seu computador em uma pasta.

## Commit

Um Commit é um pacote de alterações feitas no repositório. Cada commit possui arquivos alterados, autor e uma mensagem de resumo.

## Branches

Uma Branch no **git** é um ponteiro para as alterações feitas nos arquivos do projeto. É útil em situações nas quais você deseja adicionar um novo recurso ou corrigir um erro, gerando uma nova ramificação garantindo que o código instável não seja mesclado nos arquivos do projeto principal. Depois de concluir a atualização dos códigos da ramificação, você pode mesclar a ramificação com a principal, geralmente chamada de **Master**.

## Fork

 Fork nada mais é que uma cópia de um repositório. Esta cópia vira um clone do estado atual do repositório, fazendo assim com que você possa experimentar mudanças e feature novas sem precisar utilizar o repositório principal.

## Pull Request

Pull requests, geralmente são funcionalidades providas por Empresas de versionamento de código como **[Github](https://github.com/)** e **[Bitbucket](https://bitbucket.org/)**. Nada mais é que uma interface web amigável para você como desenvolvedor enviar/propor alterações em um repositório antes de integrar este código ao projeto oficial. Fazendo com que esta funcionalidade seja discutida previamente, e quando aprovada pela equipe que mantém o projeto, pode ser integrada ao código principal.

# Principais Comandos:

## Configuração

Basicamente, esses são os primeiros comandos que devem ser feitos. Isso porque ele irá dizer para o Git quem está fazendo as alterações do arquivo. Por exemplo, quando você realiza um commit, ele dirá quem fez, quando, etc.

**$git config --global user.name “Seu nome”**

**$git config --global user.email “seu.email@email.com”**

## Iniciar Repositório

Sempre quando for de seu desejo iniciar um repositório Git, é necessário utilizar este comando na pasta principal de seu projeto.

**$git init**

*Dica: Para saber se funcionou, basta entrar na pasta onde você realizou o comando e ver se apareceu um arquivo chamado “.git”.*

## Status

Sempre que você quiser verificar a informação do seu repositório basta utilizar o comando de *status* e se quiser um status mais resumido e com maior dinamismo, basta adicionar “-s” no final do comando.

**$git status** (Completo)

**$git status -s** (Resumido)

## Adicionar arquivos para Commit

Sempre que você quiser adicionar arquivos novo ou arquivos modificados no repositório, basta utilizar o comando *add*. Com ele você pode adicionar arquivos individuais ou até mesmo todos os arquivos do repositório.

**$git add NomeDoArquivo** (Arquivo individual)

**$git add .** (Todos os arquivos)

## Clonar Repositório

O Git clone é um comando para baixar o código-fonte existente de um repositório remoto (como o Github, por exemplo) e clonar também o arquivo .git para que possamos modificar e adicioná-lo no Repositório remoto novamente.

**$git clone URL** (URL é o link do repositório remoto)

## Mexendo com Branches

As branches, como já dito anteriormente, são ramificações do repositório. Podemos adicionar novas branches, Listar, enviar e deletar.

**$git branch NomeDaBranch** (Adiciona uma nova Branch)

**$git push –u RepositórioRemoto NomeDaBranch** (Envia a Branch para o Repositório Remoto)

**$git branch** ou **$git branch --list** (Lista todas as Branches)

**$git branch –d NomeDaBranch** (Deleta a Branch)

## Trocando de Branch

**$git checkout NomeDaBranch** (Troca para uma Branch existente)

**$git checkout –b NomeDaBranch** (Cria e troca de Branch ao mesmo tempo)

## O Commit

**$git commit –m “Mensagem para o commit”**

## Desfazendo Commits

**$git revert ‘HashDoCommit’**

## Mostrando alterações da Branch

Com o fetch, podemos ver todos os commits feitos na Branch, porém não “juntamos” com a nossa.

**$git fetch –all**

## Mesclando alterações

Com o merge podemos mesclar todas as modificações feitas de uma branch para a outra.

**$git merge NomeDaBranch**

## Pull (Fetch + Merge)

Este comando é uma combinação de git fetch e git merge, o que significa que, quando usamos o **git pull**, ele recebe as atualizações do repositório remoto **git fetch** e aplica imediatamente as alterações mais recentes no seu local **git merge**.

**$git pull RepositórioRemoto**

## Git Ignore

Ao compartilhar seu código com outras pessoas, geralmente há arquivos ou partes de seu projeto que você não deseja compartilhar.

Exemplos

* arquivos de log
* arquivos temporários
* arquivos escondidos
* arquivos pessoais
* etc.

O Git pode especificar quais arquivos ou partes do seu projeto devem ser ignorados pelo Git usando um .gitignorearquivo.

O Git não rastreará arquivos e pastas especificados em .gitignore. No entanto, o .gitignore próprio arquivo é rastreado pelo Git.

### Criando um arquivo .gitignore

**$touch .gitignore**

### Regras

|  |  |  |
| --- | --- | --- |
| **Pattern** | **Explanation/Matches** | **Examples** |
|  | Blank lines are ignored |  |
| # *text comment* | Lines starting with # are ignored |  |
| *name* | All *name* files, *name* folders, and files and folders in any *name* folder | /name.log /name/file.txt /lib/name.log |
| *name*/ | Ending with / specifies the pattern is for a folder. Matches all files and folders in any *name* folder | /name/file.txt /name/log/name.log  **no match:** /name.log |
| *name*.*file* | All files with the *name.file* | /name.file /lib/name.file |
| */name*.*file* | Starting with / specifies the pattern matches only files in the root folder | /name.file  **no match:** /lib/name.file |
| *lib/name*.*file* | Patterns specifiing files in specific folders are always realative to root (even if you do not start with / ) | /lib/name.file  **no match:** name.file /test/lib/name.file |
| \*\**/lib/name.file* | Starting with \*\* before / specifies that it matches any folder in the repository. Not just on root. | /lib/name.file /test/lib/name.file |
| \*\**/name* | All *name* folders, and files and folders in any *name* folder | /name/log.file /lib/name/log.file /name/lib/log.file |
| /lib/\*\**/name* | All *name* folders, and files and folders in any *name* folder within the lib folder. | /lib/name/log.file /lib/test/name/log.file /lib/test/ver1/name/log.file  **no match:** /name/log.file |
| \*.*file* | All files withe *.file* extention | /name.file /lib/name.file |
| \**name*/ | All folders ending with *name* | /lastname/log.file /firstname/log.file |
| *name*?.*file* | ? matches a **single** non-specific character | /names.file /name1.file  **no match:** /names1.file |
| *name*[a-z].*file* | [*range*] matches a **single** character in the specified range (in this case a character in the range of a-z, and also be numberic.) | /names.file /nameb.file  **no match:** /name1.file |
| *name*[abc].*file* | [*set*] matches a **single** character in the specified set of characters (in this case either a, b, or c) | /namea.file /nameb.file  **no match:** /names.file |
| *name*[!abc].*file* | [!*set*] matches a **single** character, **except** the ones spesified in the set of characters (in this case a, b, or c) | /names.file /namex.file  **no match:** /namesb.file |
| \*.*file* | All files withe *.file* extention | /name.file /lib/name.file |
| *name*/ !*name*/secret.log | ! specifies a negation or exception. Matches all files and folders in any *name* folder, except name/secret.log | /name/file.txt /name/log/name.log  **no match:** /name/secret.log |
| \*.*file* !*name*.file | ! specifies a negation or exception. All files withe *.file* extention, except name.file | /log.file /lastname.file  **no match:** /name.file |
| \*.*file* !*name*/\**.file* junk.\* | Adding new patterns after a negation will re-ignore a previous negated file All files withe *.file* extention, except the ones in *name* folder. Unless the file name is junk | /log.file /name/log.file  **no match:** /name/junk.file |