1 GIT [sistema de controle de versões](https://pt.wikipedia.org/wiki/Sistema_de_controle_de_vers%C3%B5es) distribuído, usado principalmente no [desenvolvimento de software](https://pt.wikipedia.org/wiki/Desenvolvimento_de_software), mas pode ser usado para registrar o histórico de edições de qualquer tipo de arquivo.

2 Característica:

**Suporte consistente para desenvolvimentos não lineares**   
suporta rápidas criações de ramos (branches) e mesclas (merges)

//e inclui ferramentas específicas para visualização e navegação de históricos de desenvolvimento não lineares.

**Desenvolvimento distribuído**Git dá a cada desenvolvedor uma cópia local completa de todo o histórico de desenvolvimento, e as mudanças são copiadas de um único repositório para outro.

//Estas mudanças são importadas como ramos (branches) adicionais de desenvolvimento, e podem sofrer uma mescla (merge) da mesma forma que um ramo de desenvolvimento local.

**Compatibilidade com protocolos/sistemas existentes**Repositórios podem ser publicados por [HTTP](https://pt.wikipedia.org/wiki/HTTP), [FTP](https://pt.wikipedia.org/wiki/FTP), [rsync](https://pt.wikipedia.org/wiki/Rsync), um protocolo Git sobre uma porta conhecida ou por [ssh](https://pt.wikipedia.org/wiki/Ssh). O Git também tem uma emulação de servidor [CVS](https://pt.wikipedia.org/wiki/CVS), o que habilita a existência de clientes [CVS](https://pt.wikipedia.org/wiki/CVS) e extensões ([plugins](https://pt.wikipedia.org/wiki/Plugins)) em diversos [ADIs](https://pt.wikipedia.org/wiki/Ambiente_de_Desenvolvimento_Integrado) a utilizar os repositórios Git.

**Manipulação eficiente de projetos extensos**Git não fica mais lento com o aumento do histórico do projeto.

**Autenticação criptográfica do histórico**O histórico do Git é salvo de uma maneira que o nome de uma determinada revisão (um "commit", ou entrega, nos termos do Git) depende de todo o histórico de desenvolvimento que leva até este commit.

// Uma vez publicado, não é possível mudar as versões antigas sem passar despercebido. A estrutura é similar a uma [árvore](https://pt.wikipedia.org/wiki/%C3%81rvore_(estrutura_de_dados)) [hash](https://pt.wikipedia.org/wiki/Hash) ([hash tree](https://pt.wikipedia.org/w/index.php?title=Hash_tree&action=edit&redlink=1)), mas com dados adicionais nos nós e nas folhas.

**Estratégias de mescla (merge) conectáveis**Git tem um conjunto bem definido de modelos de uma mescla incompleta, e possuí vários algoritimos para completá-las, culminando em comunicar ao usuário que é incapaz de completar o merge automaticamente, sendo necessária uma edição manual.

**O**[**lixo**](https://pt.wikipedia.org/wiki/Coletor_de_lixo_(inform%C3%A1tica))**se acumula se não for limpo**Abortar operações ou desfazer mudanças irá deixar objetos sem valor pendentes no banco de dados. Existe porém uma pequena fração desejável de objetos no sempre crescente histórico, mas liberar o espaço usando git gc --prune pode ser uma operação lenta.[[29]](https://pt.wikipedia.org/wiki/Git#cite_note-29)

**Empacotamento periódico explícito de objetos**O Git armazena cada novo objeto criado como um arquivo separado.

//Embora cada arquivo seja individualmente comprimido, isso requer um espaço considerável no disco e é ineficiente. Isto é resolvido com o uso de "pacotes" que armazenam um grande número de objetos em um único arquivo (ou pela rede), comprimidos pelo delta entre eles.

3 Ferramentas:

<https://www.git-tower.com/windows>

O foco do Tower é fazer com que o controle de versão seja fácil. Em uma ferramenta bonita, eficiente e poderosa. Pode-se adicionar diversos serviços como Github, Bitbucket, Gitlab, Git Flow

4 Principais comandos:

* git add <arquivos...> Este comando adiciona o(s) arquivo(s) em um lugar que chamamos de INDEX, que funciona como uma área do git no qual os arquivos possam ser enviados ao Github. É importante saber que ADD não está adicionando um arquivo novo ao repositório, mas sim dizendo que o arquivo (sendo novo ou não) está sendo preparado para entrar na próxima revisão do repositório.
* git commit -m "comentário qualquer" Este comando realiza o que chamamos de “commit”, que significa pegar todos os arquivos que estão naquele lugar INDEX que o comando addadicionou e criar uma revisão com um número e um comentário, que será vista por todos.
* git push Push (empurrar) é usado para publicar todos os seus commits para o github. Neste momento, será pedido a sua senha.
* git status Exibe o status do seu repositório atual

Merge

Branch

trenheira

fator importante do git (e essa é um dos seus diferenciais em relação ao svn – caso vc o conheça) é a possibilidade de criar, a qualquer momento, vários snapshots do seu projeto, ou como chamamos mais “nerdmenete”, branch. (seria um projeto oculto seu que voce trabalharia até tornar oficial (não seria mais oculto).

6 Vantagens dos dois:

Benefícios do Controle de Versão Distribuído

As vantagens estão relacionadas à distribuição do processamento, redundância/replicação de repositórios e às novas possibilidades de colaboração entre desenvolvedores (novos fluxos de trabalho).

### **Do Ponto de Vista do Desenvolvedor**

* **Rapidez**. As operações são processadas localmente. Não é necessário passar pela rede e contatar o servidor central para fazer um commit, log ou diff por exemplo.
* **Autonomia**. A conexão com a rede só é necessária para trocar revisões com outros repositórios. Fora isso, trabalha-se desconectado e em qualquer lugar, como num cliente por exemplo.

### **Do Ponto de Vista da Gerência/Coordenação**

Parte das decisões gerenciais envolve manter livre o caminho da equipe para que possam trabalhar da melhor maneira possível. Outras decisões importantes são sobre redução de custos. Nestes dois casos específicos, o modelo distribuído oferece as seguintes vantagens:

* **Confiabilidade**. No sistema centralizado, uma pane no servidor interrompe todo o desenvolvimento. Já no sistema distribuído, além de a equipe poder continuar seu trabalho, os repositórios dos desenvolvedores funcionam como cópias de backup de todo o projeto.
* **Redução de custos com servidor**. A carga de processamento fica distribuída nas próprias máquinas dos desenvolvedores. O repositório "central", quando existe, tem o papel do repositório "oficial" e não como processador central das requisições.

## Desvantagens do Controle de Versão Distribuído

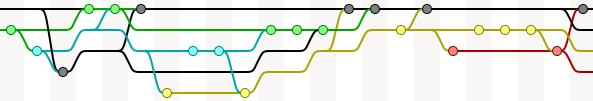
Nem tudo são flores com o DVCS.

### **Maior Complexidade**

No centralizado, a forma de trabalho é mais simples de se entender. Mesmo que limitadamente, uma pessoa com pouco conhecimento de controle de versão consegue trabalhar com o resto da equipe.

histórico do controle de versão centralizado

O DVCS é mais complicado. Há diversas combinações de arquiteturas de repositórios e fluxos de trabalho possíveis, o que pode tornar o histórico da evolução do projeto bastante confuso.



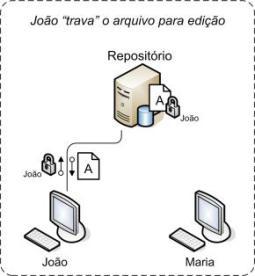
Ao contrário do centralizado, não adianta só commit e update para funcionar "no tranco". Todos os desenvolvedores da equipe precisam ter um conhecimento maior do modelo, da ferramenta e, de preferência, também de um processo de desenvolvimento que padronize fluxos de trabalho a serem seguidos.

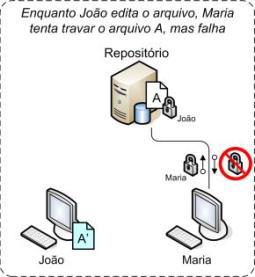
### **Travamento de Arquivos Binários Precisa Ser Centralizado**

Diferentemente dos arquivos de texto, arquivos binários possuem um formato interno que não é baseado em linhas de texto e, por isso, não podem ser mesclados automaticamente pelo controle de versão ou manualmente pelo desenvolvedor.

Sendo assim, a edição concorrente de arquivos binários é problemática. Duas pessoas editando ao mesmo tempo uma figura, por exemplo, não conseguirão mesclar as modificações depois e o trabalho de uma delas precisará ser refeito.

Com arquivos binários, a melhor solução é usar o travamento, isto é, sinalizar que o arquivo está travado para edição e que ninguém mais deve editá-lo enquanto isso.





O modelo puramente distribuído não é adequado para lidar com travamento justamente por não possuir um servidor central que possa controlar as travas de todos.

### **Controle de Mudança Ainda é Centralizado**

As ferramentas de controle de mudança (e.g. Trac, Redmine, Mantis, Bugzilla) ainda não acompanharam as de controle de versão na arquitetura peer-to-peer. Isto significa que mesmo usando um DVCS, ainda haverá uma ferramenta centralizada para controle de mudança.

## Considerações Finais

Controle de versão distribuído oferece algumas vantagens interessantes sobre o centralizado. Por outro lado, requer muito mais preparo da equipe e também do processo, o que não chega a ser realmente um impedimento uma vez que um processo formalizado e equipes capacitadas são fundamentais para produção de software de qualidade.

|  |  |
| --- | --- |
| Desenvolvedor | * Rapidez * Autonomia |
| Coordenação/Gerência | * Redução de custos com servidor e infraestrutura externa de rede * Confiabilidade |

7

1’ SVN: Apache Subversion é um [sistema de controle de versão](https://pt.wikipedia.org/wiki/Sistema_de_controle_de_vers%C3%A3o" \o "Sistema de controle de versão) open-source que gerencia arquivos e diretórios controlando as alterações realizadas ao longo do tempo desenhado especificamente para ser um substituto moderno do [CVS](https://pt.wikipedia.org/wiki/CVS" \o "CVS), O Subversion utiliza banco de dados [Berkeley BD](https://pt.wikipedia.org/wiki/Berkeley_DB" \o "Berkeley DB).

2’ Caracteristica :

* Versionamento de diretórios;
* Versionamento de renomeação, cópia e exclusão;
* Commits atômicos;
* Merge tracking;
* Bloqueio de arquivos;
* Resolução de conflitos interativos.

3’Ferramentas:

4’ Principais comandos:

* Listar os comandos do SVN:

$ svn --help

* Adicionar diretório:

$ svn add directory

* Adicionar arquivo:

$ svn add arquivo

* Commitar arquivos ou diretórios locais no repositório SVN:

$ svn commit filename

$ svn commit --message "Message" filename

* Deletar arquivo do repositório:

$ svn delete filename

$ svn delete directory

* Bloquear arquivo para garantir acesso exclusivo:

$ svn lock filename -m "comment as to why its locked or by whom"

* Mostrar as mensagens do log do SVN:

$ svn log filename

* Mostrar status das mudanças em um arquivo:

$ svn status

* Trazer todas as atualizações do repositório SVN para sua cópia local:

$ svn update

O SVN utiliza o conceito de branches, tags e trunk:

Trunk: pasta que contém os projetos que estão em desenvolvimento. Todas as atualizações efetuadas dia-a-dia são armazenadas nesta pasta.

Branches: pasta que contém “linhas de desenvolvimento” de tal projeto, que entre elas pode haver poucas diferenças, porém uma independe da outra. Quando o projeto está pronto para ser liberado como uma versão estável, a pasta trunk é copiada para a pasta branch e dado um nome de versão. Este branch é congelado, não sofrendo mais alterações, apenas correções. Os testes são efetuados.

Tags: quando os testes efetuados na branch estão completos, a versão que se encontra na branch é copiada para a pasta tags, criando a “release”. A pasta tag é empacotada e enviada para o cliente. Qualquer modificação em branch, deve ser copiada para a pasta de tags, após todos os testes. O SVN considera tag apenas uma variação de um branch, e na prática é exatamente como um branch, apenas uma cópia da ramificação atual da árvore.