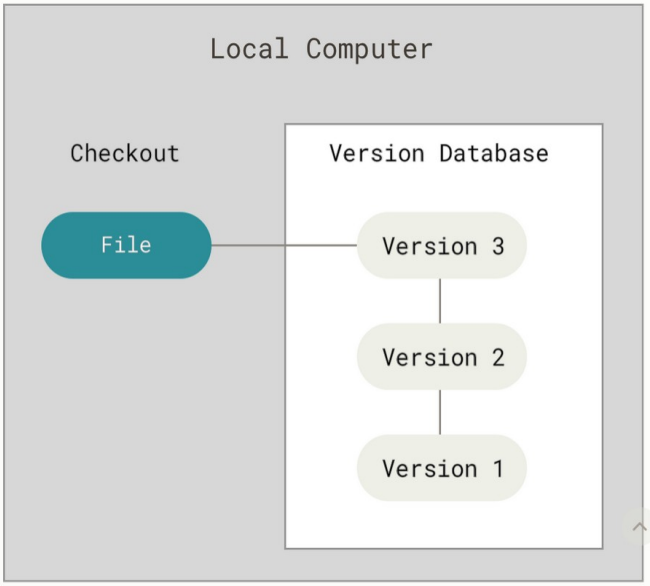
GIT

(1)INTRODUÇÃO

1 – SOBRE O CONTROLE DE VERSÃO

Controle de versão é um sistema que registra alterações em um arquivo ou conjunto de arquivos ao longo do tempo para que você possa lembrar versões específicas mais tarde, embora muito utilizado em projetos de software, podem ser usados com qualquer tipo de arquivo em um computador

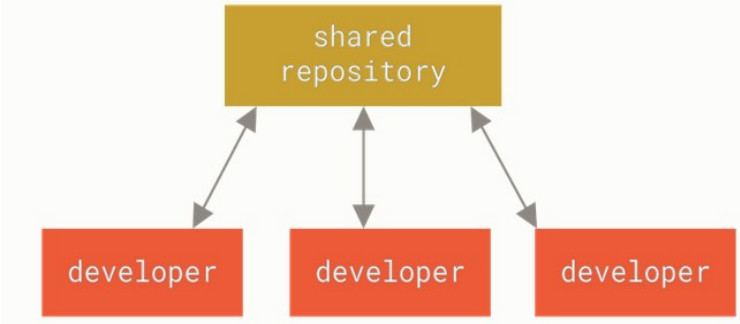
- SISTEMAS LOCAIS DE CONTROLE DE VERSÃO



O método de controle de versão de muitas pessoas é copiar os arquivos para outro diretório. Esta abordagem é muito comum porque é muito simples, mas também é incrivelmente propensa a erros.
Programadores há muito tempo desenvolveram VCSs locais que tem um banco de dados simples que mantêm todas as alterações nos arquivos sob controle de revisão.
O problema de sistemas locais de controle de versão é que, caso o disco seja corrompido e não houver um backup, todo o projeto é perdido
Uma das ferramentas VCS mais populares foi um sistema chamado RCS, que ainda é distribuído com muitos computadores hoje.

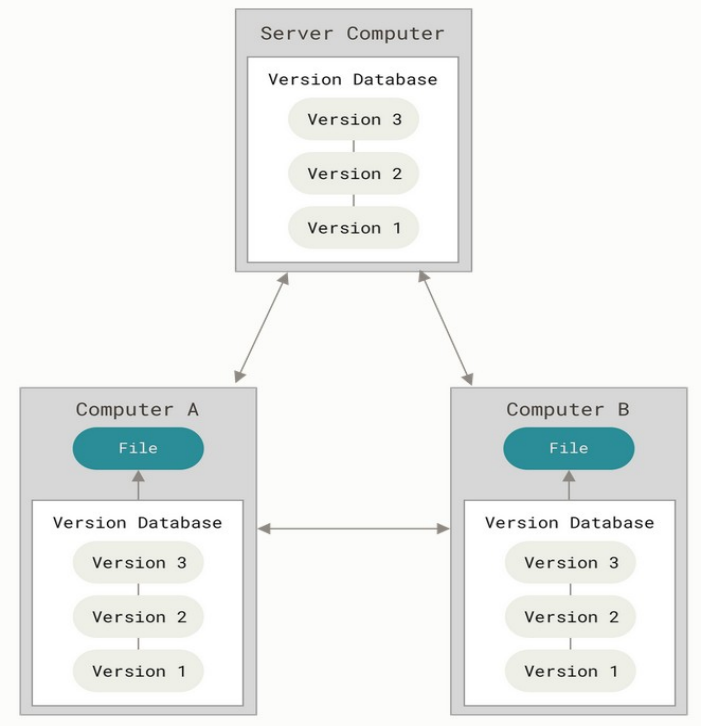
- SISTEMAS CENTRALIZADOS DE CONTROLE DE VERSÃO

CVCSs (Centralized Version Constrol Systems)foram desenvolvidos para pessoas colaborarem com desenvolvedores em outros sistemas
Estes sistemas, tais como CVS, Subversion e Perforce, têm um único servidor que contém todos os arquivos de controle de versão, e um número de clientes que usam arquivos a partir desse lugar central.



Os administradores têm controle refinado sobre quem pode fazer o que; e é muito mais fácil de administrar um CVCS do que lidar com bancos de dados locais em cada cliente.
No entanto, esta configuração também tem algumas desvantagens graves. O mais óbvio é o ponto único de falha que o servidor centralizado representa.
Se esse servidor der problema por uma hora, durante essa hora ninguém pode colaborar ou salvar as alterações de versão para o que quer que eles estejam trabalhando. Se o disco rígido do banco de dados central for corrompido, e backups apropriados não foram mantidos, você perde absolutamente tudo - toda a história do projeto, exceto imagens pontuais que desenvolvedores possam ter em suas máquinas locais.

- SISTEMAS DISTRIBUIDOS DE CONTROLE DE VERSÃO



Em um DVCS (Distributed Version Control System) (como Git, Mercurial, Bazaar ou Darcs), clientes não somente usam o estado mais recente dos arquivos: eles duplicam localmente o repositório completo.

Assim, se qualquer servidor morrer, e esses sistemas estiverem colaborando por meio dele, qualquer um dos repositórios de clientes podem ser copiado de volta para o servidor para restaurá-lo. Cada clone é de fato um backup completo de todos os dados.

Muitos desses sistemas trabalham muito bem com vários repositórios remotos, tal que você possa colaborar em diferentes grupos de pessoas de maneiras diferentes ao mesmo tempo dentro do mesmo projeto

2 – BREVE HISTÓRIA DO GIT

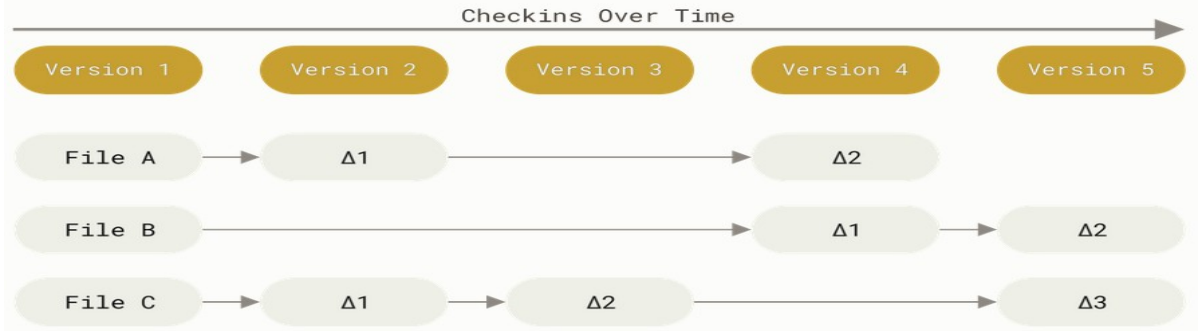
Em 2005, a relação entre a comunidade que desenvolveu o núcleo do Linux e a empresa que desenvolveu BitKeeper quebrou em pedaços, e a ferramenta passou a ser paga. Isto alertou a comunidade que desenvolvia o Linux (e especialmente Linus Torvalds) a desenvolver a sua própria ferramenta baseada em lições aprendidas ao usar o BitKeeper.
Desde seu nascimento em 2005, Git evoluiu e amadureceu para ser fácil de usar e ainda reter essas qualidades iniciais. Ele é incrivelmente rápido, é muito eficiente com projetos grandes, e ele tem um incrível sistema de **branches** para desenvolvimento não linear

3 – O BÁSICO DO GIT

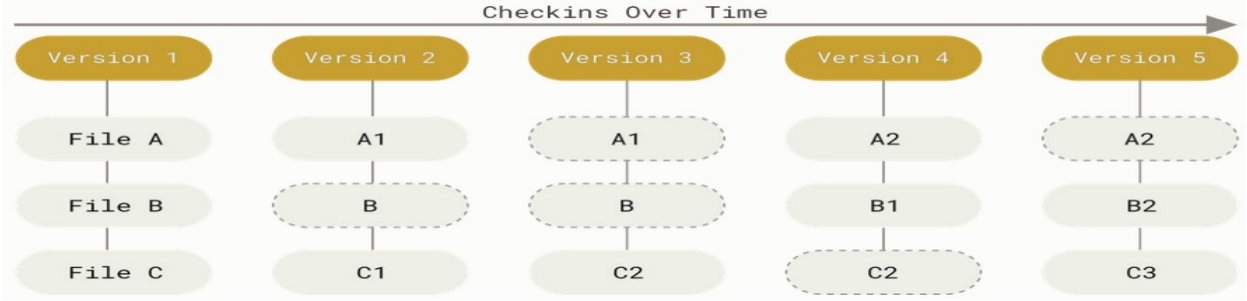
O Git armazena e vê informações de forma muito diferente do que esses outros sistemas, mesmo que a interface do usuário seja bem semelhante, e entender essas diferenças o ajudará a não ficar confuso.

- IMAGENS, NÃO DIFERENÇAS

A principal diferença entre o Git e qualquer outro VCS é a maneira como o Git trata seus dados. Conceitualmente, a maioria dos outros sistemas armazenam informação como uma lista de mudanças nos arquivos, como mostra a figura abaixo



o Git trata seus dados mais como um conjunto de imagens de um sistema de arquivos em miniatura. Toda vez que você fizer um ***commit***, ou salvar o estado de seu projeto no Git, ele basicamente tira uma foto de todos os seus arquivos e armazena uma referência para esse conjunto de arquivos. Para ser eficiente, se os arquivos não foram alterados, o Git não armazena o arquivo novamente, apenas um link para o arquivo idêntico anterior já armazenado. O Git trata seus dados mais como um **fluxo do estado dos arquivos**, como mostra a figura abaixo:



Isso faz com que o Git seja mais como um mini sistema de arquivos com algumas ferramentas incrivelmente poderosas, ao invés de simplesmente um VCS.

- QUASE TODAS AS OPERAÇÕES SÃO LOCAIS

A maioria das operações no Git só precisa de arquivos e recursos locais para operar - geralmente nenhuma informação é necessária de outro computador da rede. Como você tem toda a história do projeto ali mesmo em seu disco local, a maioria das operações parecem quase instantâneas.
Por exemplo, para pesquisar o histórico do projeto, o Git não precisa sair para o servidor para obter a história e exibi-lo para você - ele simplesmente lê diretamente do seu banco de dados local.
Isto também significa que há muito pouco que você não pode fazer se você estiver desconectado ou sem VPN. Se você estiver sem conexão à uma rede, você pode fazer ***commits*** até conseguir conexão de rede e enviar os arquivos.

- GIT TEM INTEGRIDADE

Tudo no Git passa por uma soma de verificações (***checksum***) antes de ser armazenado e é referenciado por esse ***checksum***. Isto significa que é impossível mudar o conteúdo de qualquer arquivo ou pasta sem que o Git saiba. Esta funcionalidade está incorporada no Git nos níveis mais baixos e é parte integrante de sua filosofia. Você não perderá informação durante a transferência e não receberá um arquivo corrompido sem que o Git seja capaz de detectar.
O mecanismo que o Git utiliza para esta soma de verificação é chamado um *hash* SHA-1. Esta é uma sequência de 40 caracteres composta de caracteres hexadecimais e é calculada com base no conteúdo de uma estrutura de arquivo ou diretório no Git. Um *hash* SHA-1 é algo como o seguinte:

```
24b9da6552252987aa493b52f8696cd6d3b00373
```

Conseguimos ver esses valores de *hash* em todo o lugar no Git porque ele os usa com frequência. Na verdade, o Git armazena tudo em seu banco de dados não pelo nome do arquivo, mas pelo valor de *hash* do seu conteúdo.

- O GIT GERALMENTE SOMENTE ADICIONA DADOS

Quando você faz algo no Git, quase sempre dados são adicionados no banco de dados do Git - e não removidos. É difícil fazer algo no sistema que não seja reversível ou fazê-lo apagar dados de forma alguma
Logicamente, ainda se pode perder alterações que ainda não tenham sido adicionadas em um ***commit***; mas depois de fazer o ***commit*** no Git do estado atual das alterações, é muito difícil que haja alguma perda

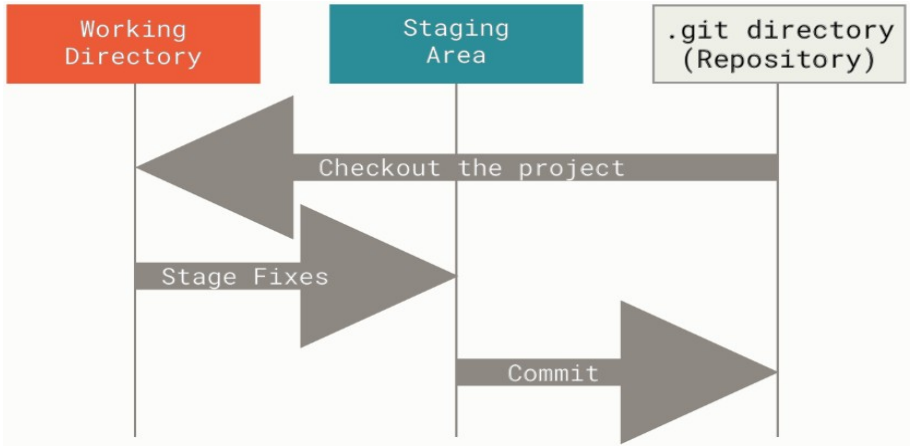
- OS TRÊS ESTADOS

O Git tem três estados principais que seus arquivos podem estar: ***committed***, modificado (***modified***) e preparado (***staged***):

- **Comitted** : Significa que os dados estão armazenados de forma segura no banco de dados local
- **Modified** : Significa que o arquivo foi alterado, mas ainda não fez o commit no banco de dados local
- **Staged** : Significa a versão atual de um aquivo modificado foi marcado como preparado para fazer parte do próximo commit

Isso nos leva a três seções principais de um projeto Git: o **diretório Git**, o **diretório de trabalho** e **área de preparo.**

- O **Git directory** é onde o Git armazena os metadados e o db de objetos de um projeto, basicamente é o que é copiando quando clonamos um repositório de outro computador
- O **Working directory** é uma simples cópia de uma versão do projeto. Esses arquivos são pegos do banco de dados compactado no diretório Git e colocados no disco para você usar ou modificar.
- A **Staged area** é um arquivo, geralmente contido no diretório Git, que armazena informações sobre o que vai entrar no próximo *commit*



O fluxo de trabalho básico Git é algo assim:

1. Você modifica arquivos no seu diretório de trabalho.
2. Você prepara os arquivos, adicionando imagens deles à sua área de preparo.
3. Você faz *commit*, o que leva os arquivos como eles estão na área de preparo e armazena essa imagens de forma permanente para o diretório do Git.

O estados serão mais explorado mais a frente.

4 – A LINHA DE COMANDO

Existem várias formas diferentes de usar o Git. Existem as ferramentas originais de linha de comando, e existem várias interfaces gráficas de usuário (GUI - Graphical User Interface) com opções variadas.
A linha de comando é o único lugar onde você pode rodar **todos** os comandos do Git - a maioria das GUI implementa somente um subconjunto das funcionalidades do Git. Se você sabe como usar o Git na linha de comando, você provavelmente descobrirá como rodar versões GUI, enquanto o oposto não é necessariamente verdade.

5 – INSTALANDO O GIT

Podemos instalá-lo como um pacote ou através de outro instalador, ou baixar o código fonte e compilá-lo.
Algumas pessoas podem achar interessante instalar Git a partir da fonte, para ter a versão mais recente. Os instaladores binários tendem a ficar um pouco atrás, embora após o Git ter amadurecido nos últimos anos, isso faz cada vez menos diferença, portanto não irei explorar a instalação da fonte aqui.

- INSTALANDO NO LINUX

Se você usar Fedora por exemplo, você pode usar o yum:

```
$ sudo yum install git-all
```

Se você usar uma distribuição baseada em Debian como o Ubuntu, use o apt-get:

```
$ sudo apt-get install git-all
```

Para mais opções de instruções de como instalar o Git em outros vários sistemas Unix, veja na página do Git, em http://git-scm.com/download/linux.

6 – CONFIGURAÇÃO INICIAL DO GIT

A configuração pode ser feita apenas uma vez por computador e o efeito se manterá após atualizações. Podemos também, mudá-las em qualquer momento rodando esses comandos novamente.
O Git vem com uma ferramenta chamada **`git config`** que permite ver e atribuir variáveis de configuração que controlam todos os aspectos de como o Git aparece e opera. Estas variáveis podem ser armazenadas em três lugares diferentes:

1. **`/etc/gitconfig`**: válido para todos os usuários no sistema e todos os seus repositórios. Se você passar a opção `--system` para `git config`, ele lê e escreve neste arquivo.
2. **`~/.gitconfig`** ou **`~/.config/git/config`**: Somente para o seu usuário. Você pode fazer o Git ler e escrever neste arquivo passando a opção `--global`.
3. **`config`** no diretório Git (ou seja, **`.git/config`**) de qualquer repositório que você esteja usando: específico para este repositório.

Cada nível sobrescreve os valores no nível anterior, ou seja, valores em **`.git/config`** prevalecem sobre **`/etc/gitconfig`**.

- IDENTIDADE

A primeira coisa que você deve fazer ao instalar Git é configurar seu nome de usuário e endereço de e-mail. Isto é importante porque cada ***commit*** usa esta informação, e ela é carimbada de forma **imutável** nos ***commits*** que você começa a criar:

```
$ git config --global user.name "Fulano de Tal"
$ git config --global user.email fulanodetal@exemplo.br
```

Você precisará fazer isso somente uma vez se tiver usado a opção **`--global`**, porque então o Git usará esta informação para qualquer coisa que você fizer naquele sistema. Se você quiser substituir essa informação com nome diferente para um projeto específico, você pode rodar o comando sem a opção **`--global`** dentro daquele projeto.

- SEU EDITOR

você pode escolher o editor de texto padrão que será chamado quando Git precisar que você entre uma mensagem. Se não for configurado, o Git usará o editor padrão, que normalmente é o Vim. Se você quiser usar um editor de texto diferente, como o Emacs, você pode fazer o seguinte:

```
$ git config --global core.editor emacs
```

- TESTANDO AS CONFIGURAÇÕES

você pode usar o comando **`git config --list`** para listar todas as configurações que o Git conseguir encontrar naquele momento:
Pode ser que algumas palavras chave apareçam mais de uma vez, porque Git lê as mesmas chaves de arquivos diferentes (**`/etc/gitconfig`** e **`~/.gitconfig`**, por exemplo). Neste caso, Git usa o último valor para cada chave única que ele vê.

Você pode também testar o que Git tem em uma chave específica digitando **`git config <key>`:**

```
$ git config user.name
Fulano de Tal
```

7 – PEDINDO AJUDA

há três formas de acessar a página do manual de ajuda (***manpage***) para qualquer um dos comandos Git:

```
$ git help <verb>
$ git <verb> --help
$ man git-<verb>
```

Por exemplo, você pode ver a *manpage* do comando config rodando

```
$ git help config
```

Caso seja necessária ajuda personalizada, você pode tentar os canais `#git` ou `#github` no servidor IRC Freenode (irc.freenode.net), ou até mesmo o GPT