Controle de Versão - O que é?

Lembra da epoca escolar onde você tinha que fazer um trabalho no word/powerpoint e toda vez você salvava com um nome diferente?

- ProjetoAgoraFInalizado.pptx
- ProjetoAgoravaiCaramba.pptx
- DEfinitivamenteMEuProjetoEStaPront.pptx

Então, se você soubesse antes o que era controle de versão, você não passaria tanta raiva.

Controle de Versão é uma maneira de gerenciar diferentes versões de um documento.

Entendendo as principais coisas:

- Responsável pelo versionamento dos documentos, sem que você precise fazer isso de maneira manual.
- Outros sistemas de controle de versões trabalham com as diferenças dos documentos, já o Git trabalha com os estados.

O que é Github?

O Github **NÃO** é a mesma coisa que o Git, ele é um serviço da web (compartilhado) que da suporte para projetos que usam versionamento com o Git.

Instalação do GIT

Acesse: https://www.hostinger.com.br/tutoriais/tutorial-do-git-basics-introducao

Configurações do GIT

- Definindo username:

```
git config --global user.name "Your Name"
```

- Definindo email:

```
git config --global user.email "Your email"
```

- Definindo editor de código padrão:

```
git config --global core.editor code (EXEMPLO)
```

- Descobrindo valor da chave:

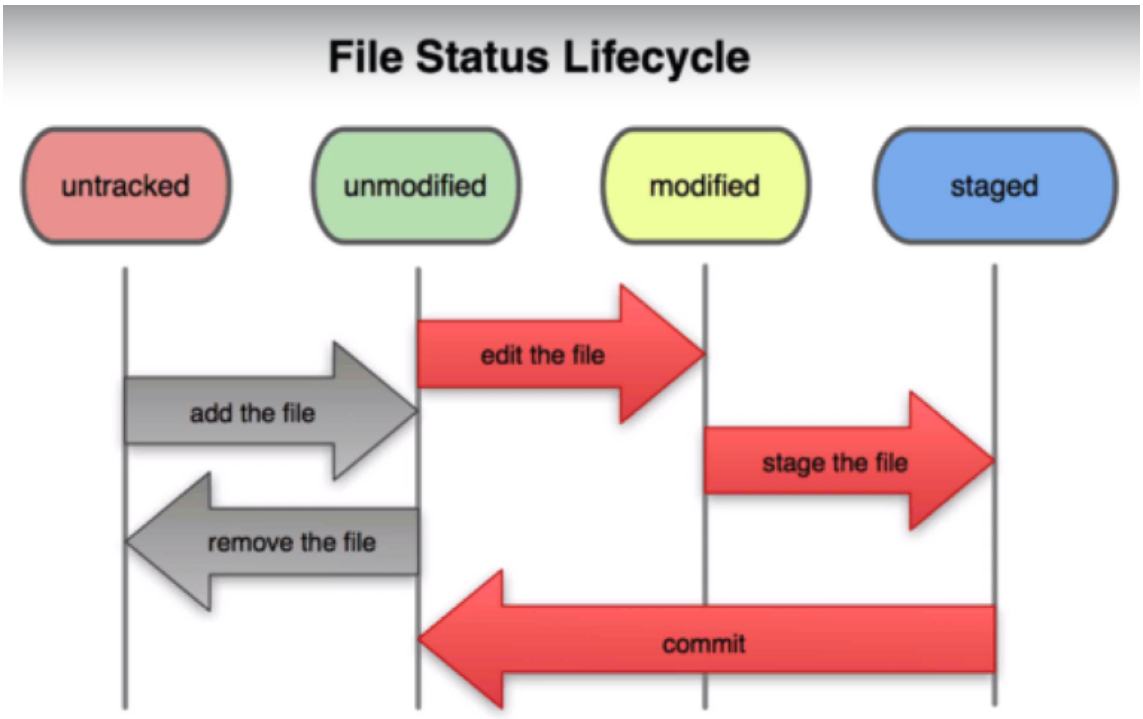
```
git config user.name
Gabriel Valin
```

Essencial do Git

Primeira coisa a se fazer com o git é inicializar um repositório, então crie a pasta do seu projeto e de um dentro dela no terminal:

```
git init
```

Ciclo de vida dos status dos Arquivos e Comandos



Comando para mostrar como está o repositório no momento:

```
git status
```

Terminal:

```
$ git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
```

Criando um arquivo:

```
vim README.md
// Escreva algo dentro dele.
```

Digite no terminal novamente:

```
git status
```

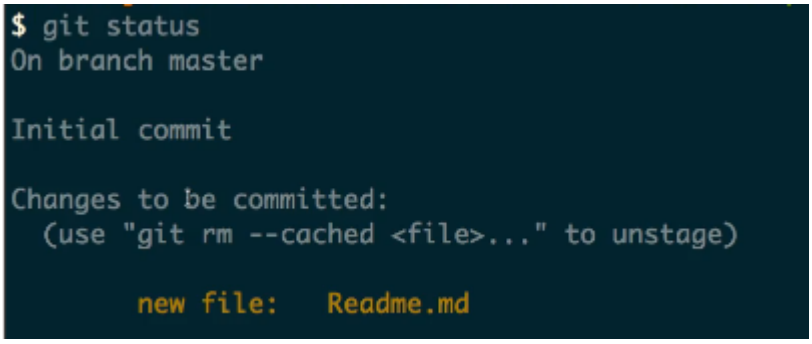
Terminal:

```
$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        Readme.md

nothing added to commit but untracked files present (use "git add" to track)
```

Podemos ver o primeiro momento do ciclo de vida. (Untracked) - Acabou de ser criado, mas o Git ainda não o reconhece.

Adicionando o arquivo e dando um status novamente:

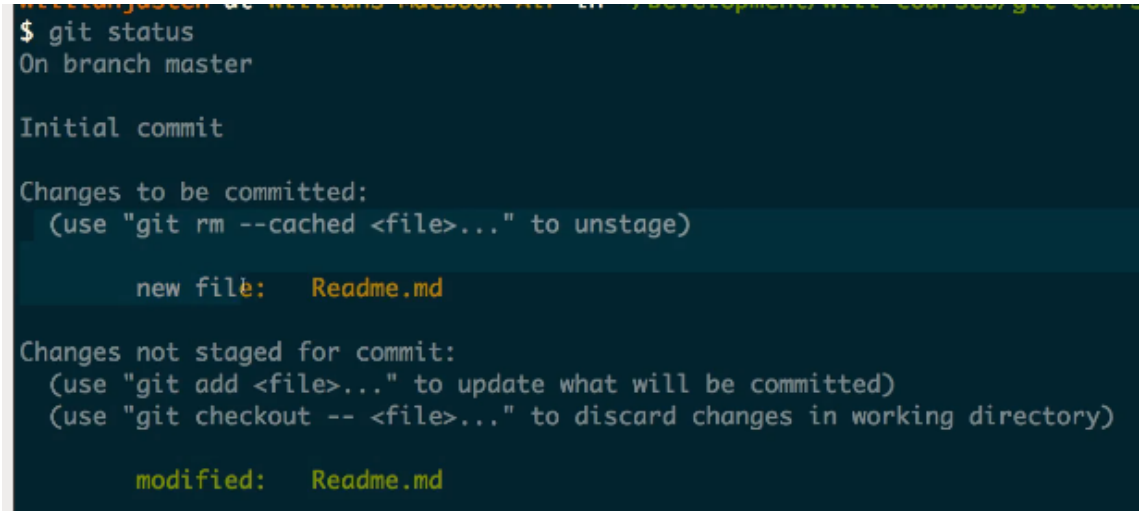
```
git add README.md
git status
```



Terminal: Podemos ver que o arquivo se encontra no estago de Staged.

Entre no arquivo README.md escreva mais alguma coisa, salve e veja o status novamente, o resultado será esse:

Terminal:



Ele está te dizendo que reconhece seu arquivo, porém ele foi modificado, se você quiser atribuir as modificações você precisa adicionar ele, utilize novamente o

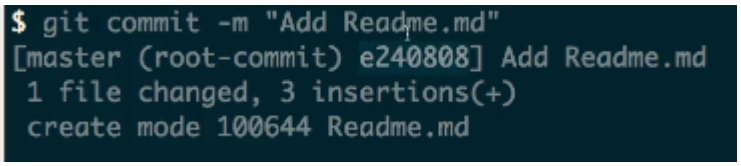
```
git add README.md
```

Commitando arquivos - Que isso?

Um commit é o momento em que você diz para o Git: "Pegue meus arquivos e crie uma versão do meu projeto!"

Como commitar?

```
git commit -m "Adicionei o README no projeto."
```



Terminal: **master = branch atual e240808 = hash do commit (IDENTIFICAÇÃO) Adicionei o README no projeto = comentário 1 file changed, 3 insertions(+) = 1 arquivo modificados com quantas linhas foram adicionadas**

Uma boa prática sempre fazer comentários semânticos e com algum sentido para o contexto quando você está trabalhando com outras pessoas, afinal, ninguém é mágico para ficar descobrindo o que você fez né?!

RECAPITULANDO: UNTRACKED = Não foi visto pelo git. UNMODIFIED = Nenhuma modificação. MODIFIED = Editado mas não foi adicionar para ser commitado. STAGED = Commitado.

Visualizando coisas importantes

Com o comando git log você pode ver coisas como os últimos commits feitos, quem fez, quando, se já foi mergeado, hash do commit, muitas coisas.. Aqui vai alguns exemplos

que podem ser interessantes:

```
git log

git log --decorate (mostra informações sobre branchs, merge, entre outras coisas...)

git log --author="Gabriel Valin" ( retorna todos os commits feitos por Gabriel Valin )

git shortlog (mostra em ordem alfabética os nomes dos autores, quantidade de commits e quais foram).

git shortlog -sn (vai mostrar quantidade de commits e quem fez)

git log --graph (mostra em forma gráfica o que está acontecendo com as branchs do repo).
```

O hash do commit é algo bem importante pois com ele conseguimos visualizar tudo que foi feito dentro dele.

```
git show <hashcommithere>
```

Diff?

Comando que permite que você veja quais alterações foram feitas antes que você envie isso para algum lugar haha, show né?!

Entre no README.me, adicione alguma linha e no terminal digite:

```
git diff
```

Ele irá te mostrar as alterações feitas no seu arquivo. (Vai que você esqueceu o que fez em algum arquivo e serve para revisar também).

Desfazendo burradas

Existe a possibilidade de você editar um arquivo salvar e ainda **não** ter adicionado e a outra é você já ter adicionado.

Solução (sem git add):

```
git checkout <filename>
```

Solução (com git add - área de STAGED):

```
git reset HEAD <filename>
// HEAD = Tirar arquivo do STAGED.
```

Dicas: o git reset possue 3 flags que podem ser passadas para ele:

```
git reset --soft (mata seu commit, mantém os arquivos modificados e volta para o STAGED pronto para ser comittado).
git reset --mixed (mata seu commit, mantém os arquivos modificados e volta para o MODIFIED)
git reset --hard (Mata o commit e tudo que foi feito nele).
```

Você sempre deve passar o hash de um commit anterior para que o git entenda a partir de qual você quer voltar!

Repositórios Remotos

Repositório remoto é aquele que fica em outro servidor, o mais conhecido é o Github.

Conectando repositório local com o remoto

```
git remote add origin <repo>
```

Enviando para repositório remoto

```
git push -u origin master

// origin = para onde
// master = de onde (local)
```

Edite o arquivo README.md, faça um commit e você poderá enviar para a branch master novamente dessa maneira:

```
git push origin master
```

Branchs

São ramificações onde diversos usuários podem trabalhar em cima de um projeto sem que ocorra alteração onde cada um está trabalhando.

- Temos a branch master (normalmente onde fica o código da produção)
- Eu recebi a tarefa de adicionar um formulário em uma página, então eu vou criar uma branch a partir da master.
- John recebeu uma outra task e também criou uma branch a partir da master para trabalhar.
- Desta maneira ambos conseguem desenvolver suas tasks sem complicações de desenvolvimento/bugs.

Criando uma branch:

```
git checkout -b newbranchname
```

Movendo e deletando branchs:

```
git branch -D newbranchname
```

Merge e Rebase

Este assunto é muito mais entendível ao se ver ele na prática, por isso recomendo que assista à estes 3 vídeos de onde estou me baseando para criar esse artigo:

Entendendo Merge: https://www.youtube.com/watch?v=R_kxAnuyQss&list=PLlAbYrWSYTiPA2iEiQ2PF_A9j__C4hi0A&index=22

Entendendo Rebase: https://www.youtube.com/watch?v=lXnRC_W2PBk&list=PLlAbYrWSYTiPA2iEiQ2PF_A9j__C4hi0A&index=23

Ambos na prática: https://www.youtube.com/watch?v=lmbwADzYJew&list=PLlAbYrWSYTiPA2iEiQ2PF_A9j__C4hi0A&index=24