Voltar uma versão anterior, alterar algo, mas p isso vc tem que fazer uma cópia e a cópia ocupa espaço

Só duplica, ele só faz uma cópia doq foi alterado (trabalha com as diferenças)

No git ele não verifica as diferenças ele tira snapshots (fotos) dos estados (trabalha com os estados)

**Github**

Serviço web compartilhado para serviços web

É um local na web que armazena os projetos versionados pelo git

Os 2 são diferentes

Github é só o local na web onde guarda o projeto

**INSTALÇÃO:**

baixar o git (https://git-scm.com/)

**CONFIGURAÇÃO:**

o git guarda suas informaçoes em 3 lugares:

- gitconfig (sistema)

- gitconfig (do sistema)

- gitconfig (do projeto)

como nos definimos em que eu quero passar pra todos repositórios vamos usar o config do usuario (global)

**Configurara user name:**

git config --global user.name "seu nome"

de um enter

**configurara o email:**

git config --global user.email "seu email"

**editor principal do git:**

git config --global core.editor "caminho ate o .exe" (C:\\Program Files (x86)\\Softwares\\Notepad++\\notepad++.exe")

**como saber o que vc colocou:**

git config user.name

**para saber tudo:**

git config --list

**INICIANDO UM REPOSITÓRIO ONDE VAMOS GUARDAR OS ARQUIVOS DE NOSSOS PROJETOS:**

**1 -** **Criar uma pasta para nosso projeto:**

mkdir nome-do-projeto

**2 -** **Entrar nela:**

cd testando-git

**3 -** **Inicializar o repositório:**

git init (inicializa o repositório e ficar enxergando todas as mudanças dentro do projeto)

o caminho para acessar o arquivo é: C:\Users\vitor e vai ter uma pasta no seu user

**- de um:** ls -la

ele vai listar os diretórios ai vc entra no .git

cd .git/

**- de um**: ls para listar algumas pastas

- Config (guarda a configuração do repositório)

- Head (qual branch padrão está)

- Breanch existente

- Descrição

- Hooks (gatilhos para certas ações)

- E outras coisas sobre objetos e referencias de cada arquivo do git

Toda vez que vc cria um repositório essas coisas vêm de padrão

**8 - USANDO TERMINAL (dá para usar notepad)**

Editar um arquivo pelo terminal do git: Vi Readme.md

- Ai vc cair em uma sessão mas vc n vai poder inserir nada, para poder fazer isso aperte **I**

- De um #Github (acho que pode escrever qualquer coisa)

- Depois de escrever tudo o que vc quer de um **ESC** (que ele vai sar do modo de inserção)

- Ai você aperta a tecla **:** que indica que você vai iniciar algum comando

- **w** (escrever e salvar)

- **q** (sair)

**9 – CICLO DE VIDA DO ESTADO DOS ARQUIVOS**

4 estados:

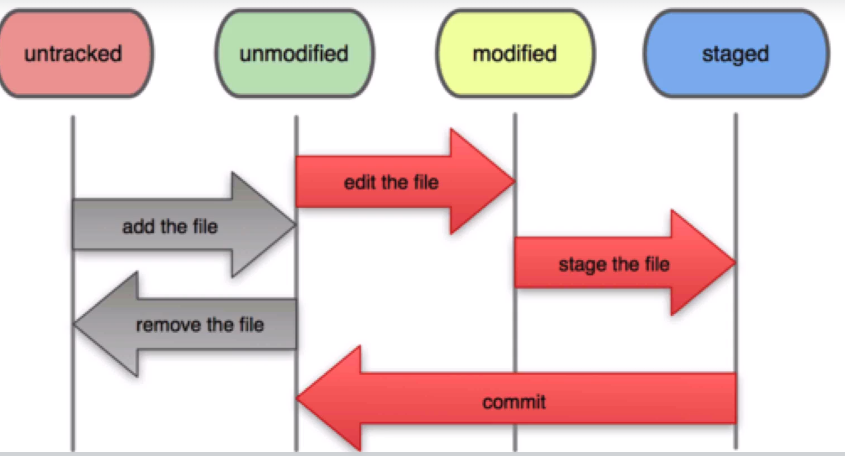
**Untracked** (não marcado): arquivo acabou de ser add no repositório, mas ainda n foi visto pelo git

**Unmodified:** depois que ele é add no git ele passa pra ca (não modificado), existe mas não teve nenhuma modificação

**Modiefied:**  se você editar o arquivo ele vem para ca (não foi add ainda)

**Staged:** depois de add, esta pronto para ser comitado

**Comitê:** criar a versão



**NO TERMINAL:**

Git status (reporta como esta o repositório)

Criando um novo arquivo:

vim Readme.md

- agora de você der um git status você vai ver que o arquivo esta na 1 etapa (Untracked acabou de ser criado mas o git não conhece ele)

- agora digite: git add nome-do-arquivo (ele vai add o arquivo)

- novamente de um git status e você vai ver que o arquivo vai estar pronto para ser comitado e versionado

- faca uma modificação no arquivo que você criou (vim nome-do-arquivo)

- Novamente de um git status e você vera que:

Changes to be committed:

(use "git rm --cached <file>..." to unstage) **EU TENHO UM ARQUIVO A SER COMITADO**

new file: Readme.md

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git restore <file>..." to discard changes in working directory)

modified: Readme.md **MAS ESSE ARQUIVO FOI MODIFICADO**

Agora você deve dar um git add novamente para preparar o arquivo para comitar.

Criando comitê (momento em que você avisa o git para pegar todos os arquivos do repositório e crie uma imagem dele, uma versão)

Git commit -m “Teste 1”(-m dizer q vc vai passar uma msg)

Git commit -am “mensagem” (-am adiciona todos os arquivos modificados mais a msg)

$ git commit -m "Teste 1"

[master (root-commit) 66e890d] Teste 1

**ELE DIZ QUE NO MASTER BRANCH Q VC ESTA ATUAL VOCE CRIOU UM COMMIT COM ESSA RASH (NUNCA VAI TER UMA VERSAO COM OS MESMOS NUMEROS) E ESSE COMENTARIO**

1 file changed, 1 insertion(+) **1 ARQUIVO MODIFICADO COM 1 LINHA**

create mode 100644 Readme.md

Se você der um git status você vai ver que não tem nada para ser feito (pq as mudanças que eu fiz já foram salvas, mas se você editar de novo e der um git status ele vai dizer que o arquivo foi modificado, mas se você não der um git add você não vai conseguir subir o arquivo porque ele esta no momento de modified mas não esta no stageded)

**10 - VISUALIZANDO LOGS**

**Git log**

- Mostra uma resh do commit (identificação)

- autor e email

- data

- msg do commit

**Git log –decorate**

- qual branch para qual brach

**Filtar por autor:**

Git log –author=”nome”

**Git shortlog:**

Mostra em ordem alfabética quais foram os autores, quantos commits e quais foram

**Quantidade de comitês e o nome:**

Git shortlog -sn

**Git log –graph**

Mostra de forma gráfica os commits

Rashs:

Por ela você sabe oq foi alterado.

Git show e-o-numero-da-rash

**11 – VER ALTERACOES ANTES DE SUBIR (antes de add)**

De um git status pra ver se tem algo em “aberto”

Edita algo

De um git diff e ele vai mostrar a modificação (isso antes do git add)

**Git diff –name-only**

Dizer somente o arquivo que foi modificado

**12 - DESFAZENDO**

**Digamos que você alterou um arquivo e quer desfazer aquilo, isso antes do commit (modified).**

Você da um: git checkout e-o-nome-do-arquivo

Fazendo isso ele vai retornar o arquivo para antes da edição

**Agora imagina que você j preparou o arquivo para o commit (ele já esta add)**

git add Readme.md

ai ele está pronto para o commit não tem mais alterações mas você esqueceu de algo

git reset HEAD(tirar o arquivo do staged(add)) e-o-nome-do-arquivo

você vai ver que ele vai sair do add e se vc der um git dif você vai ser que ainda existe a alteração

**Agora imagina que vc commitou e quer voltar:**

Git reset --soft ou –mixed ou –hard e-o-numero-de-um-comite-**antes**

**- Soft:** ele pega suas modificações e matar o commit e o arquivo já vai estar em stageded pronto para ser comitado

**- Mixed:** ele mata o commit só que ele volta os arquivo para antes do stageded (modified) se você for commitar vc vai ter que add ele ainda

**- Hard:** ele vai ignorar a existência do commit e tudo o que foi feito, vai zerar o commit, se você der um git status você vai ver que nem para modificação tem algo

**13 – CRIANDO REPOSITORIO NA NUVEM (GITHUB)**

Deixar a opção vazia

**14 – CRIANDO E ADICIONANDO UMA CHAVE SSH**

O git precisa identificar que é você quem está mandando código para aquele repositório

E essa chave SSH faz isso, ela quem faz a autenticação de um ususario remoto ao servidor

Chave privada abre a chave pública (envia a chave publica para o git hub e com a chave privada dentro da maquina abrimos a chave pública do servidor)

**1ª etapa gerar a chave:**

**- Abra o terminal e cole (sem entrar em pasta):**

ssh-keygen -t rsa -b 4096 -C "seu-email-do-github"

- Ele vai falar que gerou a chave

- Aí vc dá um enter que é uma parte em que ele fala que vai criar uma chave no caminho que ele já pré-seta

- De enter duas vezes

- Chave criada.

**- Diretório onde fica as chaves do SSH:**

Cd ~/.ssh

- de um ls que ele vai listar as chaves

- id\_rsa.pub essa é a chave para colocar no git hub

- Para pegar a chave você usa o comando:

cat e-o-nome-da-chave ou more ou abrir no editor de texto

- Então ele irá te mostrar a chave

- Copie a chave va no github

**2ª etapa adicioná-la no github**

- vá até configurações da sua conta e você verá a opção Chaves SSH e GPG

**15 - LIGANDO REPOSITÓRIO LOCAL A UM REMOTO**

Entre na pasta e cole esse código:

git remote add origin https://github.com/vitorcesar019/think.git

ele vai add um repositório remoto com o nome origin(nome default para determinar a origem do remoto, mas pode ser qualquer um) depois o endereço do repositório do git utilizando SSH

- Se você der um:

git remote ele vai listar os repositórios remotos

- De um:

Git remote -v ele tras mais info

- Agora cole esse comando:

**git push -u origin master**

ele envia os arquivos que você tem tudo (log, modificações) para o repositório em que vc está determinando.

Ele vem do branch master (padrão quando vc começa no git) e vai para o origin

**-u** quer dizer que da próxima vez vc n precisa degitar dnv “origin master”

**16 - ENVIANDO MUDANÇAS PARA UM REPOSITÓRIO REMOTO**

Digamos que você alterou algo e commitou, mas lá no Github n vai direto então você tem que fazer isso:

Git push origin (onde eu quero enviar, repositório remoto) e-o-branch-em-q-vc-esta

**17 - CLONANDO REPOSITÓRIOS REMOTOS**

Git clone permite você clonar todo um repositório para sua maquina local

Obs: você deve sair do seu repositório (pasta)

Git clone e cola o código copiado

**18 - FAZENDO FORK DE UM PROJETO**

Você pega o projeto de alguém faz alguma alteração pq achou que estava errado e manda para pessoa dona do código

Ele é diferente do clone porque o clone você pode ate alterar e baixar só que você n consegue enviar de volta

**19 - O QUE É UM BRANCH E POR QUE USAR**

Ponteiro móvel que leva a um commit

Porque usar?

Vantagens:

- Modificar arquivos sem alterar o local principal (master) (corrigir um bug enquanto outras pessoas estão trabalhando)

- Facilmente desligavel (criar e apagar branchs), ele é diferente do VCS que você subia para o git hub e demorava um pouco

- Múltiplas pessoas trabalhando

- evito de conflitos

**20 - CRIANDO UM BRANCH**

Git checkout -b(criar branch) e-o-nome-dle

Git branch

ele te da dos brachs e qual vc esta (\*)

**21 - MOVENDO E DELETANDO BRANCHES**

- Liste os branchs:

Git branch

- ir para outro branch:

Git checkout e-o-nome-do-branch

Apagando branch que eu não quero mais utilizar:

Git branch -d e-o-nome-do-branch

**22 - Entendendo o Merge**

União dos branches (branch externo ao master)

**Merge:**

Pro:

- Operação não destrutiva (não vai destruir commit, ele vai criar um novo commit e juntar tudo)

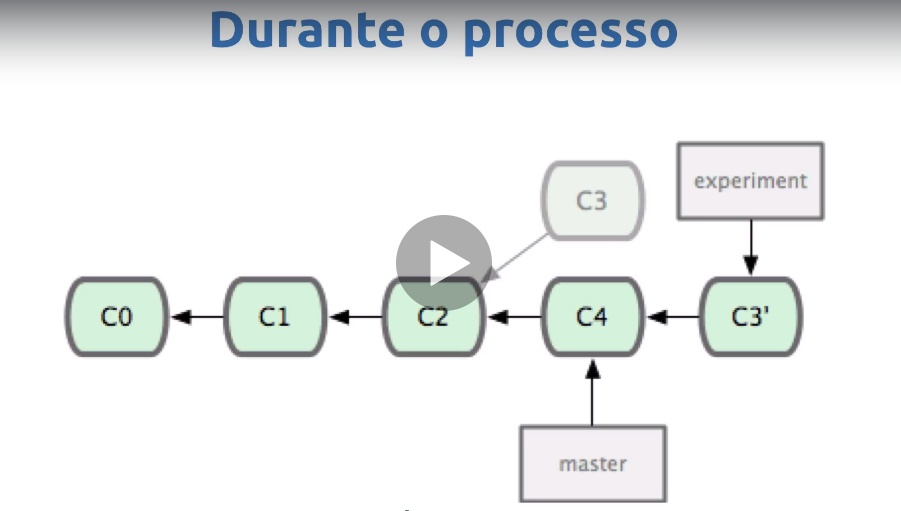
Contra:

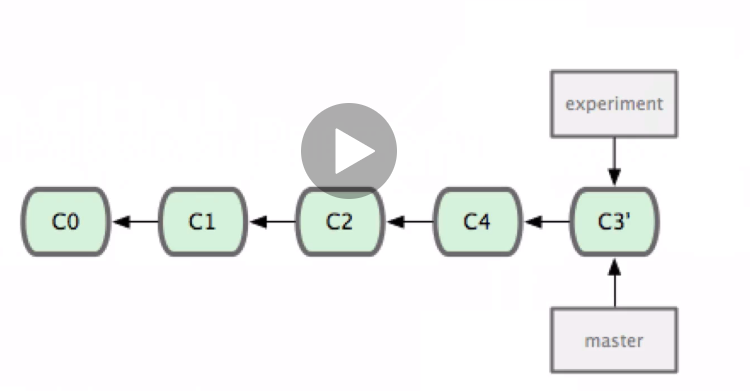
- Commit extra (1 commit só para união)

- Histórico poluído

**Rebase:**

ele pega o commit e move para frente de onde ele está jogando (pode ser o master no caso) assim tudo fica linear (pega tudo do branche separado e bota no início da fila, fastforworts, todas mudanças para frente da fila), você não cria uma arvore tendo um ciclo é tudo retilíneo.





Pro:

- Evita commits extra

- Histórico linear

Contras:

- Perda de ordem cronológica

É bom fazer o rabase em um brache sozinho ou fazer tudo de uma vez, pq se uma pessoa for fazer um commit depois de ter feito um commit vai dar ruim.

**24 - Merge e Rebase na prática**

Criando um novo repositório:

- Entre na sua pasta de usuário

- Digite: **mkdir rebase-merge**

- Entre no repositório: **cd rebase-merge/**

- Inicialize o repositório: **git init**

- Crie um arquivo: vim foo

- De um git status para ver como está o arquivo

- De um git add no arquivo **git add foo**

- commit o arquivo: **git commit -m "add foo"**

ESSE COMMIT FOI FEITO NO BRANCH MASTER

Criando um novo branch:

- git checkout -b test

- crie um arquivo: vim bar

- agora de um add no arquivo: git add bar

- commit ele agr: git commit -m “Add bar”

De um git log e você vai perceber 2 commits 1 do master e um do test

- Voltando para o master: git checkout master

- De um git log e você vai ver que so tem o commit Foo o Bar não

- crie mais um arquivo: vim fizz

- add ele

- commit ele

- Agora no git loog do master vai aparecer o foo e o fiz e não tem o bar (ele entraria no meio dos 2, de acordo com o histórico de add, ele foi feito antes do fizz)

**USANDO O MERGE NESSE CASO:**

Git merge test

Ele vai abrir uma tela, e vai falar que vai mesclar o branch test e dentro de test tem um commit (Bar)  
- de um ESC e digite :wq

- e pronto, ele add o bar para dentro do master

- de um git log e você ira perceber que o histórico ficou linear (NÃO A ESTRUTURA), foo – bar – fizz mas teve um novo commit so para fazer essa junção

- No git log –graph você percebe isso

**AGORA O REBASE:**

Crie um arquivo: vim buzz (no master

Add ele e commita

Agora vamos criar um novo branch:

Git checkout -b rebase

- nele cirei um nome arquivo bla

- add e commita ele

Vai para o master agora

- de um git log

- você vai ver que não tem o bla (que é do outro branch (rebase))

- crie outro arquivo

- add e commita

- de um git log

Você vai ver que existe o buzz e o seila no branch MASTER e no meio e para ter aquele commit **BLA**

Se fosse um merge ele iria criar um novo commit e iria juntar os 2

- no rebase:

Git rebase rebase

- ele fala que primeiro ele vai ta aplicando a modificação para o topo da lista

- se você der um log no master você vai ver que ele esta em primeiro e quebrou a ordem do histórico

- de um git log –graph

Ele juntou, e você não teve que criar um novo ciclo, então a estrutura ficou linear e não precisou de um commit extra

O MERGE É USADO NO PULL REQUEST QUE VOCE PRECISA VER QUE FOI UNIDO, PQ NO REBASE SO ENTRA UM COMMIT E VOCE NÃO SABE SE ELE VEIO DE OUTRO BRANCH, NO MERGE VOCE VE.

**25. CRIANDO O .GITIGNORE**

Gitignore serve para ignorar para não subir alguns arquivos que você deseja, você não quer subir mas precisa no seu ambiente.

Ele é um arquivo no diretório principal do repositório e dentro dele você escreve alguns padrões para ele não utilizar.

No seu repositório de um ls para listar seus arquivos

- na raiz do repositório: vim .gitignore

- agora da pra vc dizer oq não quer (da para especificar o arquivo e todas as extensões):

\*.json (tudo oq tem .json)

- de um git status e você ira perceber que os .json não vao estar disponíveis

- se você tirar a linha (apertando dd) os json fica disponível

- agora vamos ingnorar 1 arquivo so:

db.xls

Se você entrar no repositório no github de github barra gitignore la tem a base para alguns projetos, bloqueando alguns arquivos