Git e GitHub for Beginners

O que vou aprender?

1. Controle de versões;
2. Principais comandos do Git;
3. Como subir os códigos para o GitHub.

1. Controle de Versões

- **O que é? O Git** é um sistema com finalidade de gerenciar as diferentes versões de um documento. Logo pode-se avançar e voltar versões dos arquivos sem dificuldades. Perceba que ele guarda os estados do arquivo na figura abaixo.

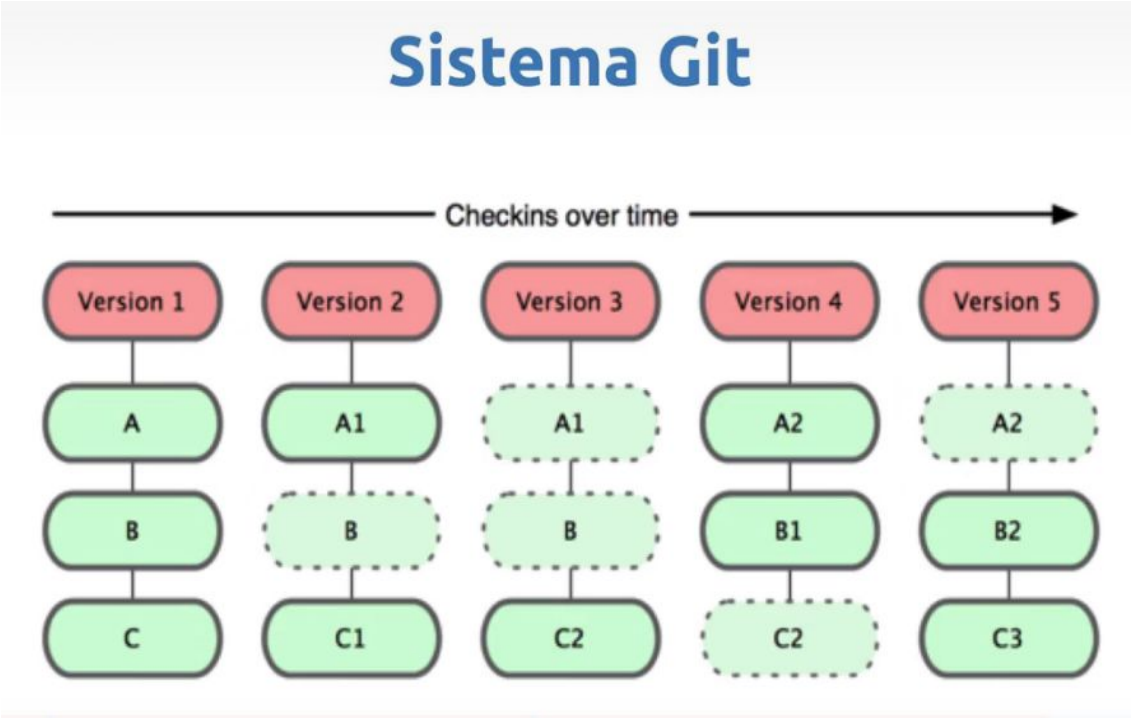


Figura 1 – Controle de Versões do sistema Git.

- **Motivações de ter isso?**
    1. Cópia e mais cópias de um mesmo projeto;
        a. Isso ocasiona em maior consumo de memória.
    2. Apagar arquivos importantes que não teriam como recuperar.

- **Dois pontos principais no Git**
    1. Versionar os arquivos do projeto, o qual não é preciso fazer isso manualmente;
    2. Trabalha com os estados dos arquivos, diferente de outros sistemas que trabalham com as diferenças deles;

Caso queira ver uma breve história do Git:
https://git-scm.com/book/pt-br/v1/Primeiros-passos-Uma-Breve-Hist%C3%B3ria-do-Git

Figura 2 – Representação dos arquivos impossíveis de se recuperar.

O que é o Github?

Algumas pessoas podem pensar que Git e Github são coisas iguais, mas não são!

O Github é um serviço de web compartilhado para projetos que utilizam o Git para versionamento, ou seja, ***Git e Github são coisas diferentes***. Basicamente, o Github é um local na nuvem (cloud) que armazena as versões de projetos do Git. Já o Git faz basicamente o controle de versões dos projetos.

Figura 3 - Git e Github são diferentes.

Download do Git

Para o download do Git acesse o site deles < https://git-scm.com/downloads > e escolha seu sistema operacional e siga as instruções dadas por eles.

Configurações do Git

- Username: **git config --global user.name “seu_user_name”**
- Email: **git config --global user.email “seu_email”**
- Editor principal: **git config --global core.editor seu_editor** (o padrão é o emacs, mas pode colocar o sublime por exemplo)

Para saber essas informações basta usar os comandos:
git config user.name
git config user.email
git config --list

Criando e configurando o repositório no Git

- Criando a pasta/diretório: **mkdir nome_pasta**
- Entrar na pasta: **cd nome_pasta**
- Inicializar o repositório na pasta que está: **git init** (Um diretório chamado .git será criado)

Usar o editor no terminal

Pode-se usar um outro editor de texto qualquer, como por exemplo: gedit, notepad++, sublime, visual studio code, eclipse e afins. Na maioria das vezes sempre vai usar o editor de texto e usar o terminal só para a parte do Git.

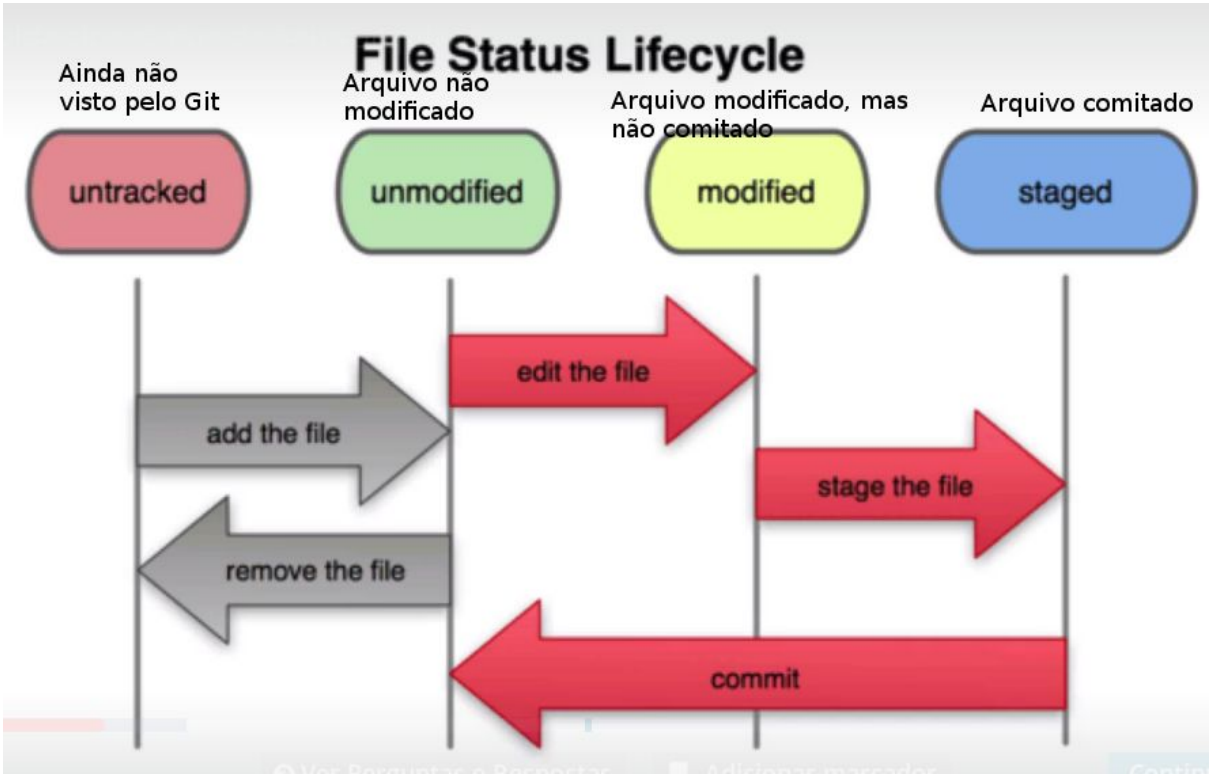


Figura 4 - Estados dos arquivos do repositório.

git status -> checa os estados/status dos arquivos do repositório, ou seja, se já foram inclusos ou não, se estão monitorados e afins
git add nome_arquivo -> adiciona o arquivo para poder comitar. Ou seja, só poder fazer commit se o(s) arquivo(s) estão adicionados.
git add . -> Adiciona todos os arquivos de uma vez.
git commit -m "sua mensagem. Por favor coloque o que realmente fez" -> criar uma imagem/snapshot dos arquivos do repositório
git commit -am "sua mensagem. Por favor coloque o que realmente fez" -> Faz tipo o conjunto de add com o commit. Lembrando que adiciona todos os arquivos modificados. Entretanto não funciona se o arquivo é untracked (não monitorado) ainda.

Visualizar os logs (He's ur friend for sure. Trust him. xD)

git log -> Mostra quem foi o autor dos commits realizados do repositório, assim como a data, horário, as mensagens dos commits e o seu hash de identificação.
git log --decorate -> Mostra informações a mais, como de qual branch para qual branch, quais tags geradas e afins.
git log --author="nome_autor" -> Faz uma consulta filtrando o autor, ou seja, todos os commits realizados por ele, caso exista.
git shortlog -> É um tipo de logo pequeno dizendo o nome, onde ele mostra de ordem alfabética os autores dos commits, assim como a quantidade de commits e quais foram.

git shortlog -sn -> Mostra os nomes das pessoas e a quantidade de commits realizadas por cada uma.
git log --graph -> Mostra de forma gráficas o que está acontecendo com os branchs e versões.
git show hash_do_commit -> Mostra quais modificações foram realizadas no commit realizado através da identificação da hash do commit.

Visualizando o diff

git diff -> Mostra as modificações realizadas antes de commitar, ou seja, antes do staged também. Muito importante para verificar se foi feito algo errado antes de commitar.
git diff --name-only -> Mostra somente o nome do arquivo que foi modificado.

Desfazendo coisas/ Resetar erros

É possível fazer modificações de erros realizados anteriormente. Em versões anteriores. Abaixo segue como se pode fazer isso.
git checkout nome_do_arquivo -> Volta o arquivo para antes da edição, ou seja, para antes de tu ter feito a "cagada". Para checar insira o comando **git diff** e verá que não há modificações no arquivo que foi modificado.
git reset HEAD nome_do_arquivo -> Diferente do comando acima ele vai tirar o arquivo do estado staged, ou seja, voltar ao arquivo depois que foi realizado o comando **git add**.
git reset --soft hash_do_commit_desejado -> Mata o commit realizado, porém voltando com o arquivo no estado de staged.
git reset --mixed hash_do_commit_desejado -> Mata o commit realizado, porém voltando para o estado modified ou seja, antes do staged.
git reset --hard hash_do_commit_desejado -> Mata o commit realizado e também sua existência por completo. Deve ser usado com cuidado (be careful son LUL). Ainda mais se os arquivos já estiverem no repositório web, no caso, github.

Github

Até agora foi utilizado somente o repositório local, no caso com o uso do Git. O Github dá a possibilidade de utilizar um repositório remoto, que fica no servidor (nuvem ou outra máquina). Primeiro entre no site do Github e para criar um novo repositório procure um + no canto superior direito e new repository.

Criando e Adicionando uma chave SSH

Essa chave basicamente serve como uma identificação para saber que você é realmente o usuário daquele repositório remoto, assim ele utiliza o SSH. Segundo o Github ele é:

“Using the SSH protocol, you can connect and authenticate to remote servers and services. With SSH keys, you can connect to GitHub without supplying your username or password at each visit.”

Todas as instruções estão no link a seguir < https://help.github.com/articles/connecting-to-github-with-ssh/ >

1. Clique em Generating a new SSH Key and adding it to the ssh-agent e siga as instruções de lá;
2. Abra a pasta oculta .ssh pelo terminal, que geralmente é criada na pasta raiz (~);
3. Use o comando **cat id_rsa.pub** ou **more id_rsa.pub** para pegar a chave de identificação e copie a chave;
4. Logo após no Github vá em: settings -> SSH and GPG keys -> New SSH key -> Cole a chave no campo Key e coloque um título bom em Title, como por exemplo o nome da máquina ou local que tu acesse essa máquina -> Add SSH Key.

Agora pode subir os arquivos/códigos sem problema de identificação do Git para o Github da determinada máquina registrada.

Ligando repositório local ao remoto

Antes de tudo crie um repositório local como já foi mostrado acima, em que usa o comando **git init**. Também pode seguir as instruções dadas na página inicial de cada repositório criado no Github. Lá ele dá as instruções para criar um repositório local, caso já esteja criado como ligar e também de importar códigos de outros repositórios. Vamos partir da ideia que já foi criado o repositório local e precisa conectar com o remoto.

git remote add origin git@github.com:endereco_do_repositorio -> Conecta com o Github, repositório remoto.
git remote -> Dentro do repositório local após ter usado o comando de conexão com o repositório remoto ele mostra que já existe um repositório origin.
git remote -v -> Mais info. do que o comando acima.
git push -u origin master -> Envia todos os arquivos, logs e afins do repositório local para o remoto.
git push origin master -> Envia as modificações de todos os arquivos feitos no repositório local.
git pull origin master -> Pega as informações do repositório remoto e atualiza no local.
OBS.: o origin do push e pull é um nome padrão dado. Você pode dar o nome que quiser. Também no comando do pull só precisa digitar **git pull**.

Clonando repositórios remotos

Na página inicial de algum repositório remoto clique em clone e copiar o link SSH. Insira os seguintes comandos.

git clone link_copiado_do_repositorio_local nome_pasta_dado_por_voce

Muito bom, muito útil e utilizado, pois caso você tenha mais de uma máquina ou está trabalhando e precisa da cópia.

Fazendo Fork

Basicamente pega um projeto que não é seu e faz uma cópia dele, ou seja, como se fosse uma cópia de maneira a contribuir com o dono do repositório. Quando fazer as alterações e dar o push para o repositório remoto o master/dono do repositório receberá requests aceitando ou não as modificações realizadas.

Perceba que Fork é diferente do clone, porque no clone tu só pega uma cópia do documento e fica para tu todas as modificações realizadas e não consegue subir para o Github. Já o Fork serve mais como uma cópia para ajudar/contribuir no projeto e você consegue subir para o Github. Lembrando que o Fork a cópia é feita no seu próprio Github e pode visualizá-lo como um repositório.

Branches (Ramos)

Um branch é um ponteiro móvel que leva um commit, como por exemplo o branch padrão chamada master que sempre aponta para o último commit criado. Entretanto isso pode ser modificado. Mas basicamente branches servem para apontar para commits existentes do repositório criado.

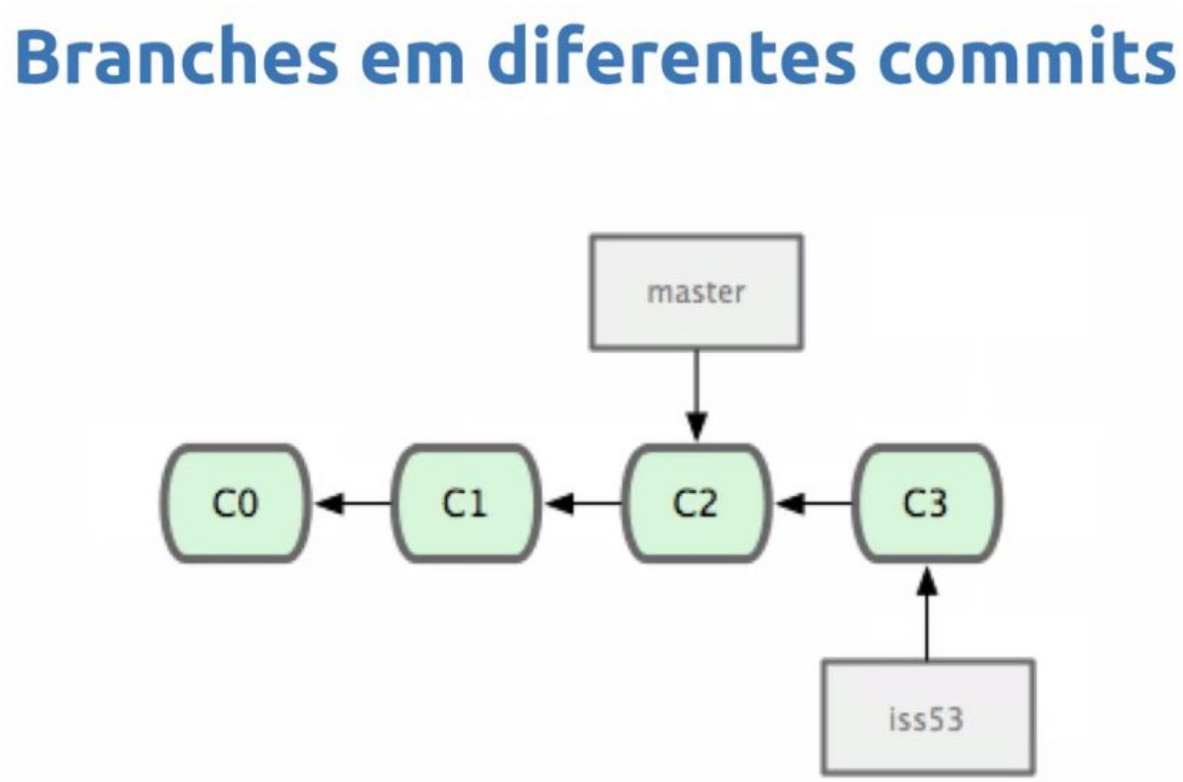


Figura 5 - Branches nos diferentes commits criados.

Vantagens

- Alterações sem modificar o local principal (branch master), ou seja, pode-se trabalhar em outras branches sem mexer na master;
- Pode-se criar e deletar branches com facilidade, tornando-o facilmente "desligável", o que é bem mais rápido que em outros sistemas;
- Várias pessoas trabalhando no mesmo projeto ao mesmo tempo sem atrapalhar uma a outra, pois podem trabalhar em branches diferentes;
- Evita conflitos, pois como dito anteriormente, os indivíduos podem trabalhar em diferentes branches o que evita conflitos, pois cada um trabalha de forma separada. Mas aí um pergunta? se todos trabalham separados como fazer para juntar todas alterações de todos os branches? basta usar uma técnica do Git chamada merge ou rebase, que junta as branches com a branch master com todas as alterações realizadas.

Comandos para o branch

git branch -> Mostra todos os branches do repositório e em qual branch você está.
git checkout -b nome_do_branch -> Cria um novo branch e muda para ele.
git checkout nome_do_branch -> Muda para o branch desejado, caso ele exista.

git branch -D nome_do_branch -> Deleta o branch desejado, caso exista. Lembrando que não pode deletar um branch que você está usando.

Unindo os branches

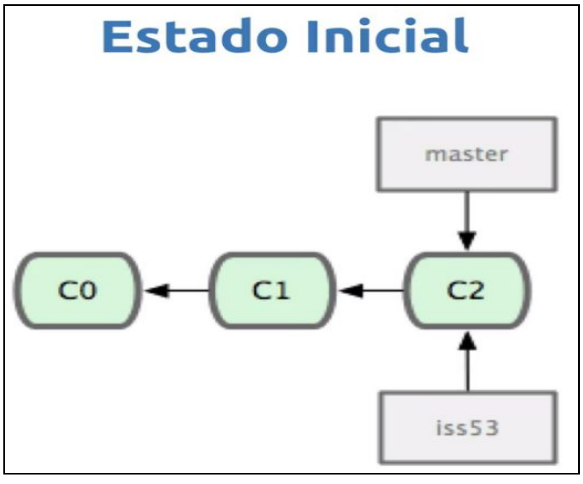
Essa parte é muito importante porque depois que foram feitas as alterações em cada branche externo é necessário fazer a união com o branch principal (master). Nisso usam-se os métodos merge e rebase.

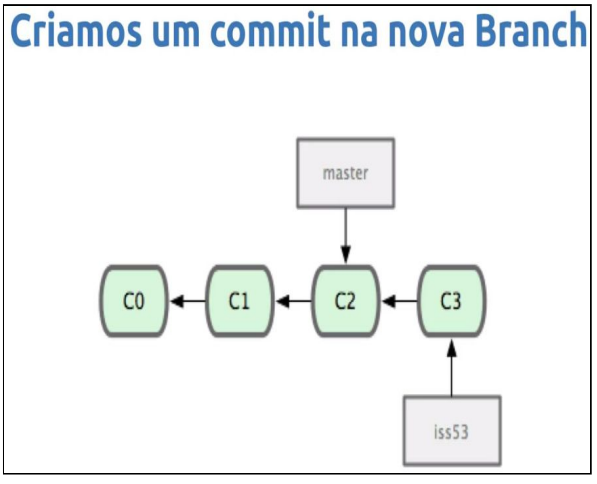
Merge

Basicamente nele, as ramificações criadas por outras branches são unificadas por um branch extra controlado pelo branch master, ou seja, é necessário criar um commit novo só para juntar todas alterações realizadas em todos os branchs externos. A vantagem dele está na não destruição/modificação no histórico dos branchs, ou seja, fica tudo registrado.

Figura 6 - Prós e contras do Merge.

Abaixo estão figuras com as etapas do merge. Não está explicado porque é muito teórico e complicado de explicar sem um esquema.





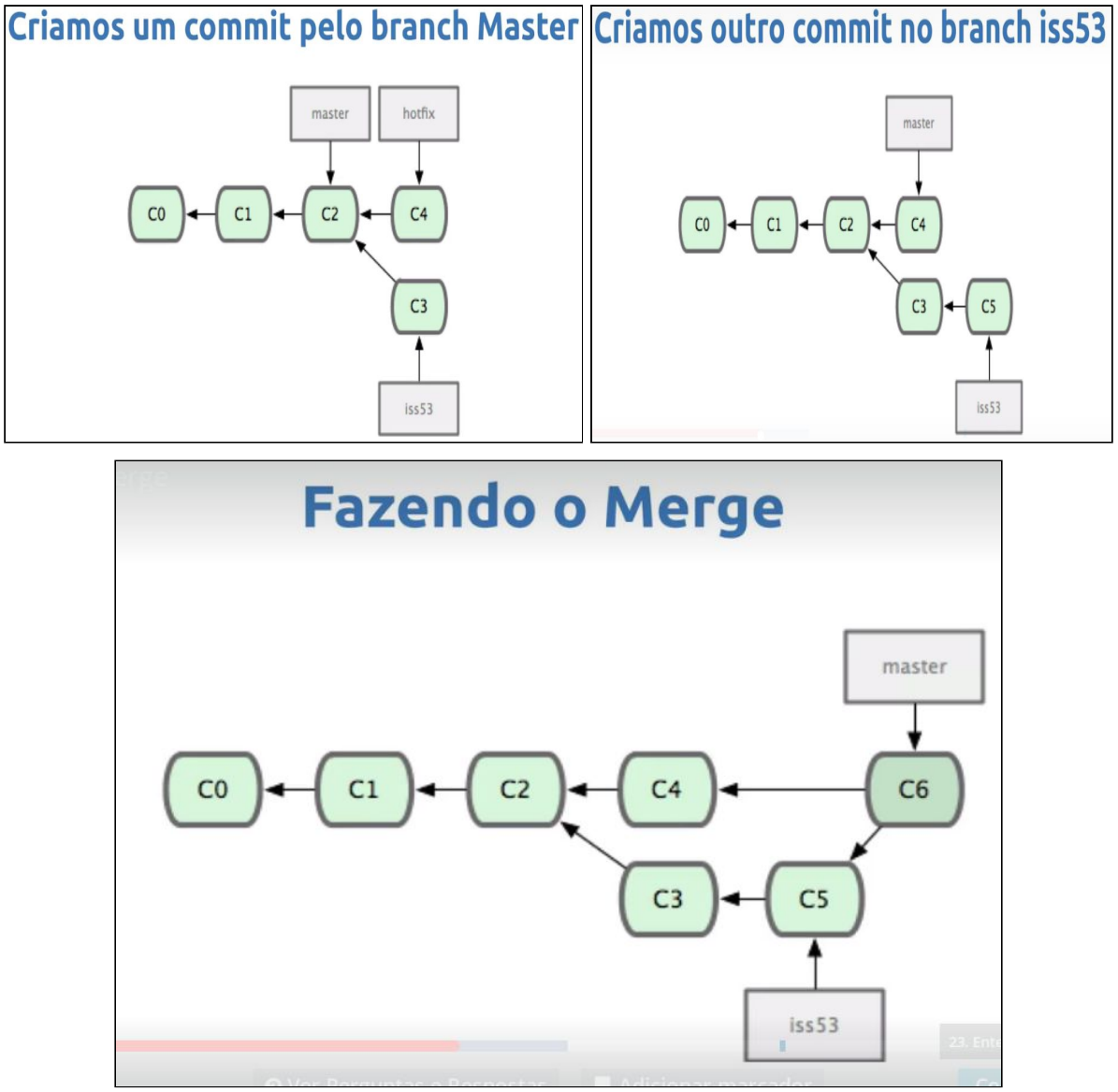
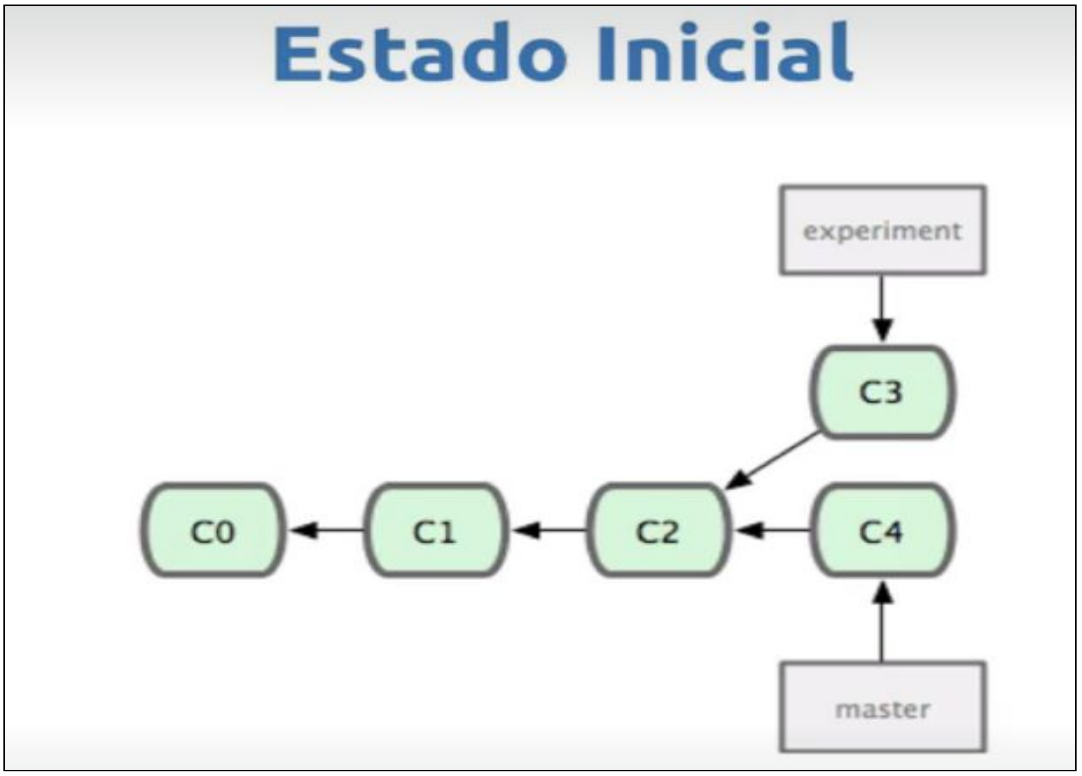


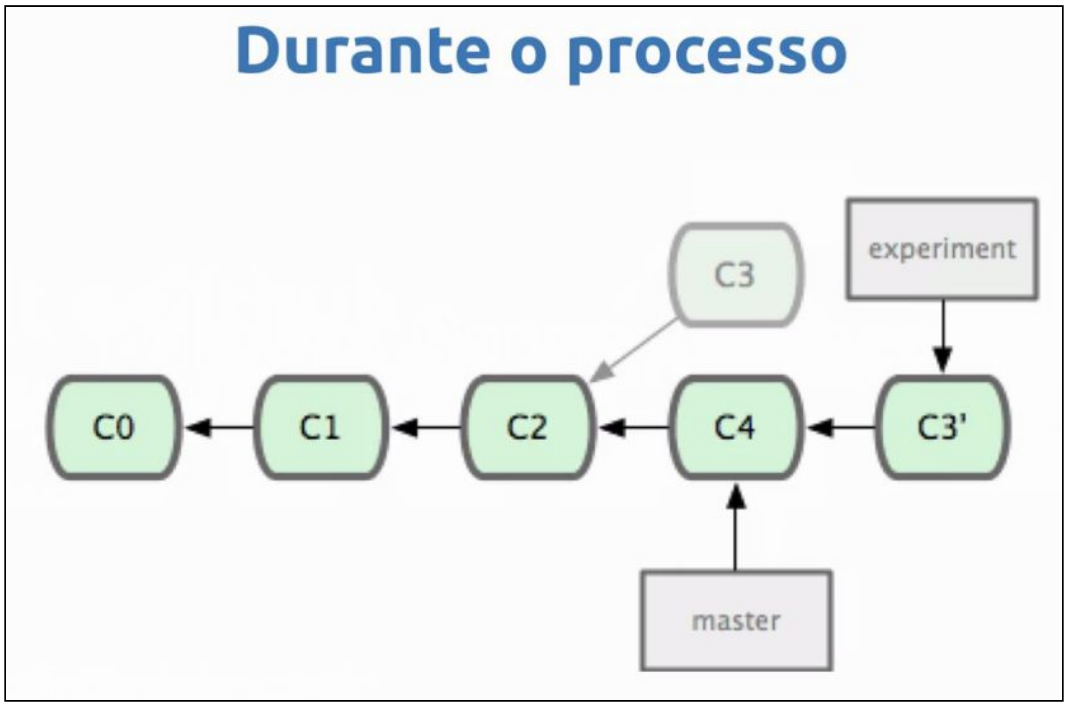
Figura 7 - Processo do Merge.

Rebase

Diferente do Merge, o Rebase unifica os branches de forma que fique linear, ou seja, não criando uma árvore com ciclo como o Merge faz. Ao fim todos os branches apontam para o mesmo commit, evitando commits extras. O bom disso é a facilidade em saber a origem dos commits e não confundir como no Merge, criando um histórico linear. Entretanto há a perda da ordem cronológica dos commits e informações do histórico.

Figura 8 - Prós e contras do Rebase.





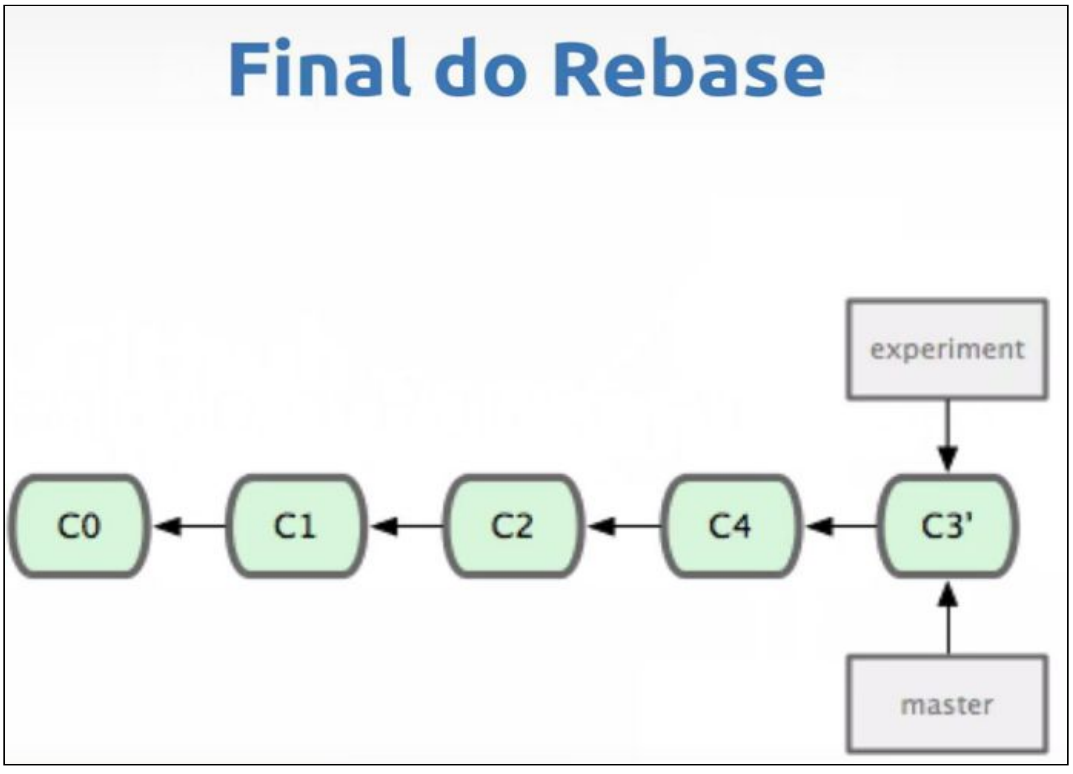


Figura 9 - Processo do Rebase.

Comandos do Merge e Rebase
Lembrando que fazer explicação de exemplos práticos é ruim. Então procure exemplos na internet. Quando for fazer na prática é bem simples, só tem que entender. Abaixo só estão alguns comandos dos métodos Merge e Rebase.

git merge nome_do_branch -> Faz a unificação entre os branches de todos os commits realizados.
git rebase nome_do_branch -> Faz a unificação entre os branches, porém diferente do merge como mostrado anteriormente.

git log --graph -> Ele é interessante de ser usado porque conseguirá ver forma gráfica como estão se comportando os branchs e as unificação deles é mostrada quando se usa ou o Merge ou o Rebase.

OBS.: Geralmente é sempre melhor usar o rebase do que o merge, principalmente quando se usa o git log --graph ou git log para saber os históricos de commits realizados. Por quê? o motivo é que o histórico fica muito poluído e muitas das vezes quando temos muitos branches fica uma zona ler com o merge. Ele é realmente necessário quando se deseja checar todo o histórico incluindo as junções.

Extras

Gitignore
Basicamente serve para não subir arquivos de forma intencional, muitas vezes ligadas por motivos de segurança. Para fazer isso é necessário criar um arquivo do tipo .gitignore e nele colocar os arquivos e suas extensões que não quer que sejam “trackiados”, ou seja, aderido ao Git.

1. Crie um arquivo tipo **.gitignore** no repositório com vários arquivos;
2. Abra esse arquivo e digite os nomes dos arquivos, um embaixo do outro (\n) que deseja inibir/bloquear/ignorar o acesso. Dica: caso deseje ignorar todos os arquivos de uma dada extensão use *.extensao_desejada;
3. Por fim salve o arquivo e verifique pelo terminal com o comando **git status** e verá que ele irá ignorar os arquivos do passo 2.

Para mais informações sobre como funciona exatamente o gitignore acesse o site com a documentação < https://git-scm.com/docs/gitignore >.
Também caso queira templates de padrões de ignore para determinados tipos de projetos. Bem legal checar ele e usá-lo quando necessário. Segue o link < https://github.com/github/gitignore >.

Comandos muito importantes que quase ninguém usa no dia a dia como por exemplos:

git stash -> Guarda modificações ainda não commitadas em uma pasta e depois chamar quando achar necessário. Geralmente usado quando faz aquelas modificações, mas ainda não acabou. Vá no repositório e digite esse comando e verá que o arquivo vai para o modo WIP (Work In Process).

git stash apply -> Retorna com os arquivos que estavam sendo modificados que foram colocados em modo WIP. Lembrando que quando é feito o apply os arquivos no stash continuam lá, onde caso queira deletar deve usar o comando git stash clear que está logo abaixo.

git stash list -> Mostra todos os arquivos no stash daquele repositório.

git stash clear -> Limpa todo o stash juntamente com os arquivos.

OBS.: **git stash** é muito mais muito útil quando vai fazer um **git pull** de um repositório de um repositório remoto e não quer que tenha conflitos e problemas com isso.

git config --global alias.seu_atalho comando_git -> Esse comando vem a partir do **Alias**, que basicamente serve para fazer atalhos de comandos para agilizar o processo de digitalização. Lembrando que essa configuração será global do seu git.

git config --local alias.seu_atalho comando_git -> Faz exatamente mesma coisa do comando acima, porém não é global e sim local somente do repositório que você se encontra.

git tag -a numero_versao -m "Mensagem desejada que faça sentido" -> Basicamente servem como marcadores dos commits de um repositório, remoto principalmente. Por exemplo, supomos que acabou um parte do projeto e quer marcar até esse ponto de finalização, como se fosse um versionamento (1.0, 2.0 e etc). Logo crie uma tag e suba para o github (remoto) com o comando **git push origin master --tags** e cheque a aba de release da página inicial do repositório.

git tag -> Mostra todas as tags criadas.

git tag -d nome_da_tag -> Apaga a tag criada e inclusive o nome da tag é a versão que você deu a ela.

git revert hash_do_commit -> Vai fazer algo parecido com o comando **git reset**, porém ele não apaga o commit que deseja ser revertido. Ele apenas volta ao commit anterior, mas deixando registrado o commit que foi revertido. Isso muitas das vezes serve para deixar registrado algum erro realizado no código em um projeto.

OBS.: Sempre que deletar os branches ou as tags no repositório local não irá ser apagado no repositório remoto. Para fazer isso basta usar os seguintes comandos:

git push nome_repositorio_remoto(padrão é origin) :nome_da_tag(versão da a ela)

git push nome_repositorio_remoto(padrão é origin) :nome_do_branch

Made by Harã Heique and based on course named as: Git e Github para iniciantes