TRYBE
Modulo I - Introdução ao Desenvolvimento Web

Bloco 2 – Git e GitHub, Internet

1) Git e GitHub

Definições
GIT = Sistema de controle de versão distribuido (ao invés de centralizado que requer lock),
GITHUB = Local onde hospeda seu git na web.

Fluxo

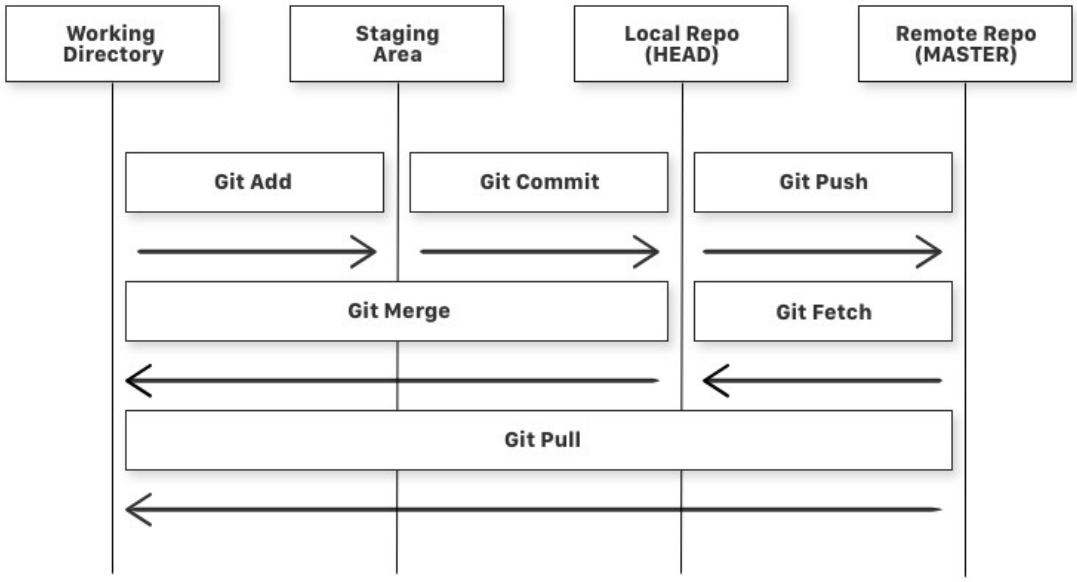
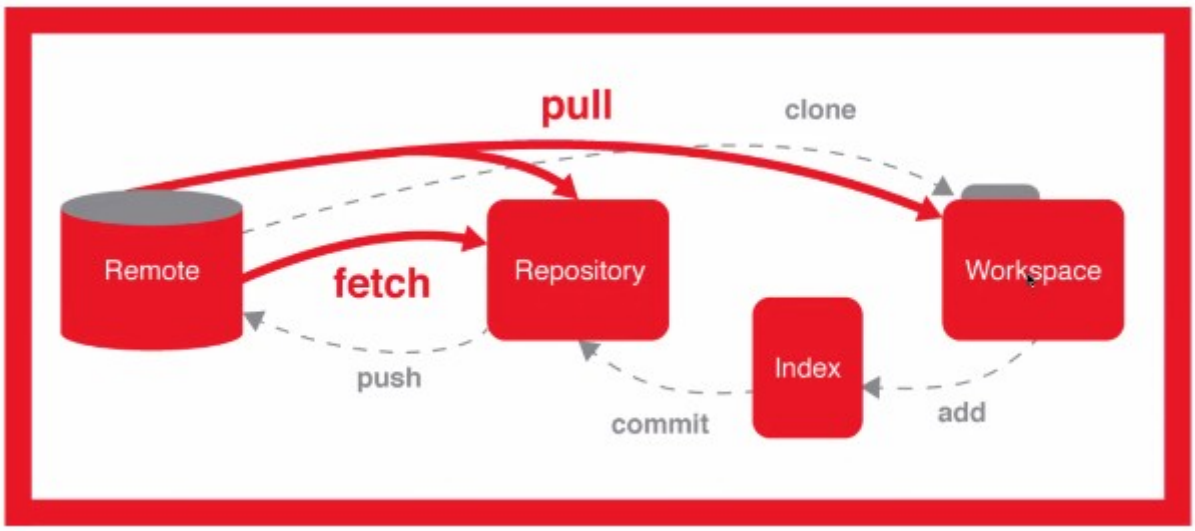
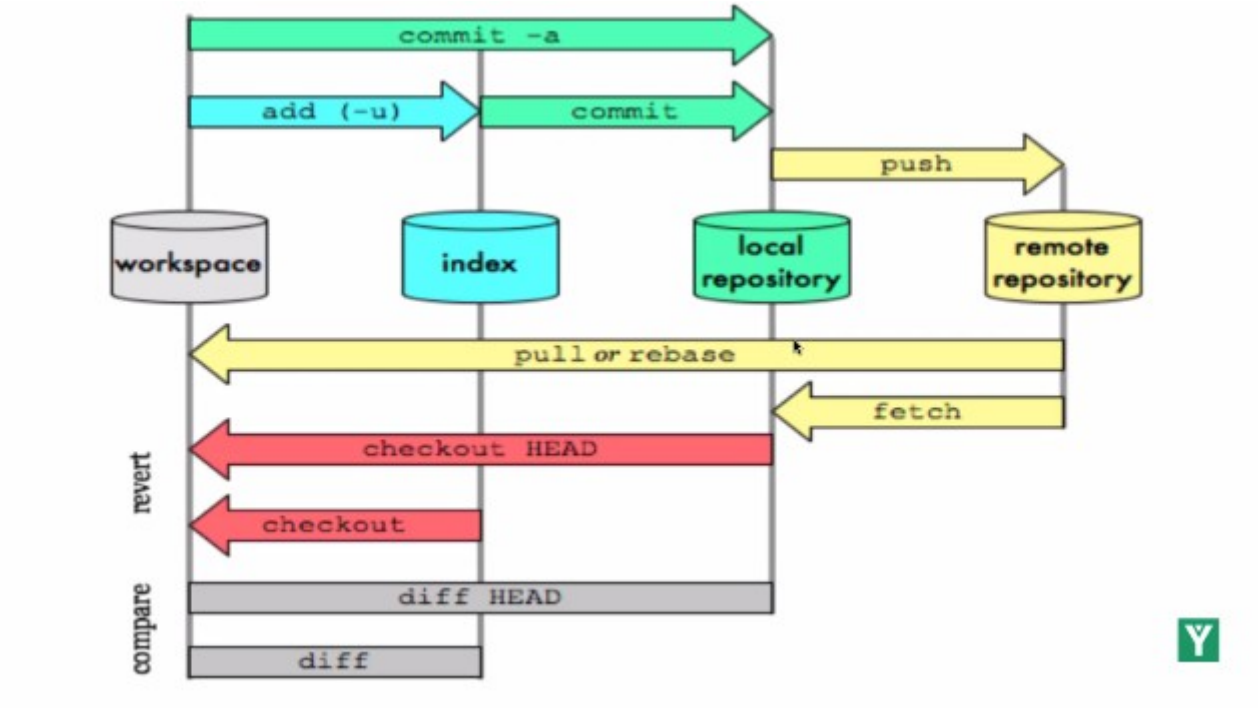


Diagram of a simple Git Workflow



(onde repository é repositorio local e remote é repositorio remoto)



Diferentes exemplos de fluxos

<u>- Criar um documento e subir no seu repositório</u>

*Criar um repositório localmente, git init
*Adicionar (add) e comitar (commit)
*Criar um repositório no próprio GitHub
*Conectar o repositório local com o repositório público
*Sincronizar ambos os repositórios

<u>- Clonar, modificar e subir no repositorio</u>

*Mesmo fluxo mas iniciando com clone e sem git init
*cuidado com commits bem na branch (e no final, deletar ela)
*conectar com git remote add origin opcional segundo em que repo quer push
*push
*no final, git pull desde a master, se quiser refazer o ciclo sem erro

<u>Entender o pull request</u>

Objetivo: colaborar, fazer sugestões nas alterações de um repo. Sugeridas na branch, aprovadas para passar na master. 2 cenários: **tem autorização sobre repo?** SIM (**clone e cria branch** para sua pull request) ou NÃO (**bifurca** repo primeiro via fork).

Comandos mais úteis

```
$ git config --global user.name "John Doe"
```

```
$ git config --global user.email johndoe@example.com
```

To check config, ex: `git config user.name`

--
git branch – verificar em que branch
git branch namebranch – criar nova branch
git checkout branch – entrar na branch
git checkout -b namebranch - Criar e entrar de uma vez na branch
git checkout – navegar entre versoes dos arquivos
--
git log – hitorico de commits (pode ser git log nomeespecifico)
git show namecommit – ver commit especifico (git show mostra ultimo por padrao)
--
git rm namefile – remover
git checkout refcommit~1 namefile – restaurar
--
merge - juntar branch feature e branch master
--
git add
git ignore (pegar arquivo .gitignore no site https://gitignore.io/)
git commit -m “”
--
git remote add origin urldestination (to connect rour git and github repo)
git remote set-url origin nameofnewurl (to correct access, between HTTP and SHH = Secure Schell)
git remote rm origin (delete a remote)
git remote -v (check url names and status)
ls -a (to see everything including .git)
rm -fr .git (to delete git repo)
--
git push origin namebranch
git push -u origin namebranch (primeira vez, para depois git push, ou se ainda não conhecido)

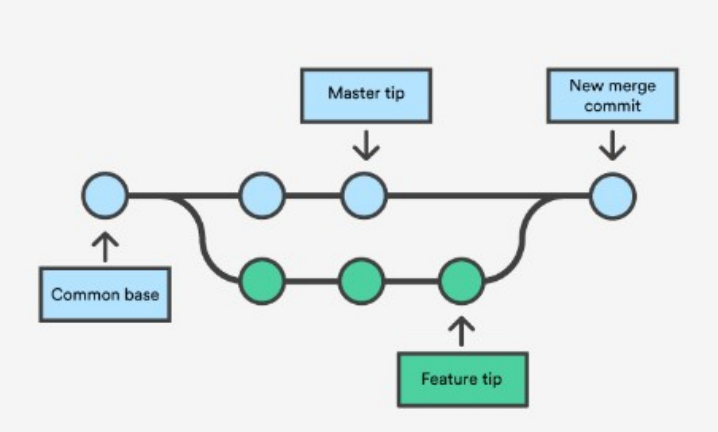
git fetch origin / git pull origin (**pull = fetch + merge** pois pull é para pegar alteração e já fazer o merge, fetch apenas pega a alteração)
--
Cuidados: com o `rebase` e `reflog`

Referências: material integral oficial de comandos do Github, pdf salvo.

Git merge

Para unificar um histórico bifurcado.
(Associado com comandos como git checkout e git branch -d.)

Fluxo:

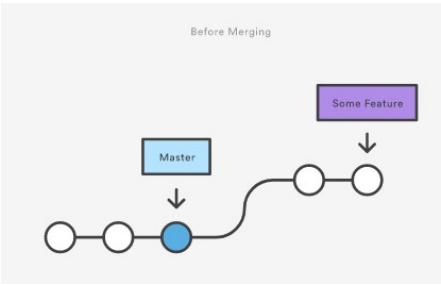


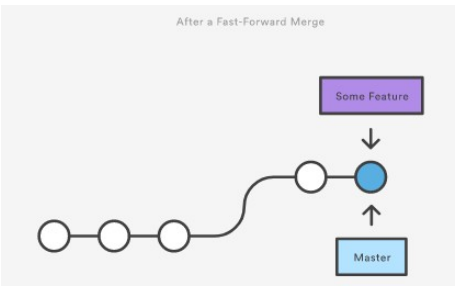
Fonte: https://www.atlassian.com/br/git/tutorials/using-branches/git-merge

Dicas para dar certo:
- confirmar o branch receptor com *git status* (verificar onde aponta o head). Caso não for certo, executar *git checkout <receiving>* , ou seja geralmente é *git checkout master*
- buscar os commits remotos mais recentes, executando *git fetch* e uma vez confirmado, *git pull.*
- etapa de mesclagem: *git merge <branch name>* .

Dois modos principais de mesclagem

→ Mesclagem de avanço rápido
Mais eficiente, quando existe um caminho linear do branch atual até o branch alvo. Impossível se as branch tiverem divergido, ou seja não saim do mesmo commit.
Ex de uso: resolvendo bugs.

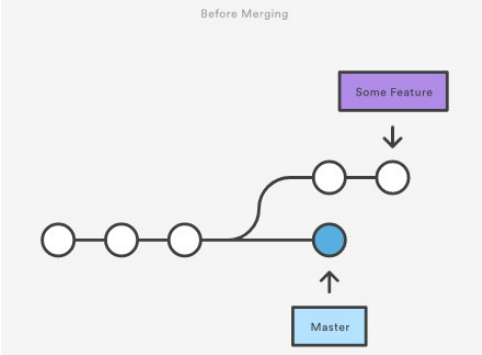


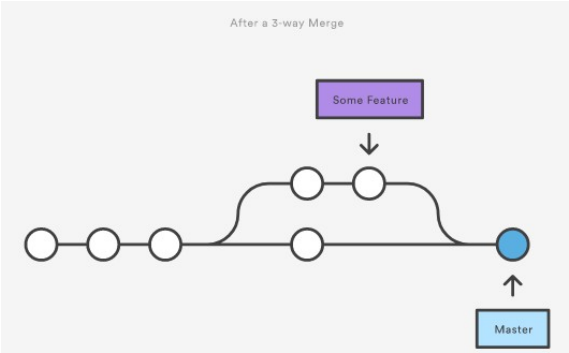



```
# Iniciar um novo recurso
git checkout -b new-feature master
# Editar alguns arquivos
git add <file>
git commit -m "Start a feature"
# Editar alguns arquivos
git add <file>
git commit -m "Finish a feature"
# Mesclar com a branch do novo recurso
git checkout master
git merge new-feature
git branch -d new-feature
```

DICA:
git merge --no-ff <branch>
(gera commit de mesclagem, útil para documentar todas mesclagens do repo)

→ **Mesclagem de três vias**
Ex: recursos grandes, muitos desenvolvedores no mesmo projeto.






```
Start a new feature
git checkout -b new-feature master
# Editar alguns arquivos
git add <file>
git commit -m "Start a feature"
# Editar alguns arquivos
git add <file>
git commit -m "Finish a feature"
# Desenvolver a branch principal
git checkout master
# Editar alguns arquivos
git add <file>
git commit -m "Make some super-stable changes to master"
# Mesclar com a branch do novo recurso
git merge new-feature
git branch -d new-feature
```

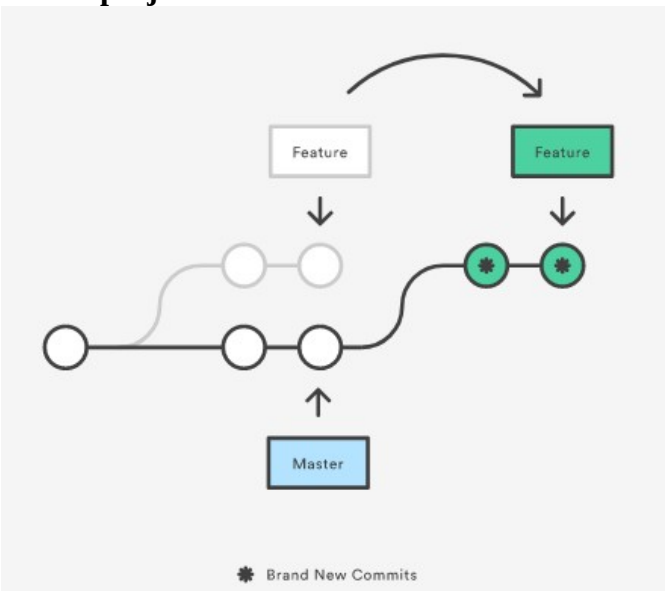
Opcional: pode fazer **rebase** para recolocar na minha linha.

Conflitos
Fluxo de trabalho edit/stage/commit já sinaliza. Em caso de conflito, o Git não faz merge porque não pode escolher a versão certa. Executar git *status* para detetar qual parte foi alterada mais de uma vez.
Git mostre conflito com <<<<<<<, =======, e >>>>>>>.

Merge no PR padrão do Github
Usa sempre o *no fast forward (--no-ff)* ou seja gera commit de merge, boa prática de rastreabilidade.

Git rebase

Processo de mover ou combinar uma sequência de confirmações para uma nova confirmação base.
Para **manter um histórico de projeto linear.**



Como funciona
git rebase master : atualiza minha branch com o que tem na master, criando um histórico comum. Analiza commit por commit e faz merge. Em caso de conflito dá alerta.

Cuidados de uso
- usar apenas quando branch não pública, sem ninguém mais trabalhando, pois muda o histórico.
- Usar apenas com workflow bem claro, entendido e definido.

Referência:
https://www.atlassian.com/br/git/tutorials/rewriting-history/git-rebase .

Mais dicas concretas

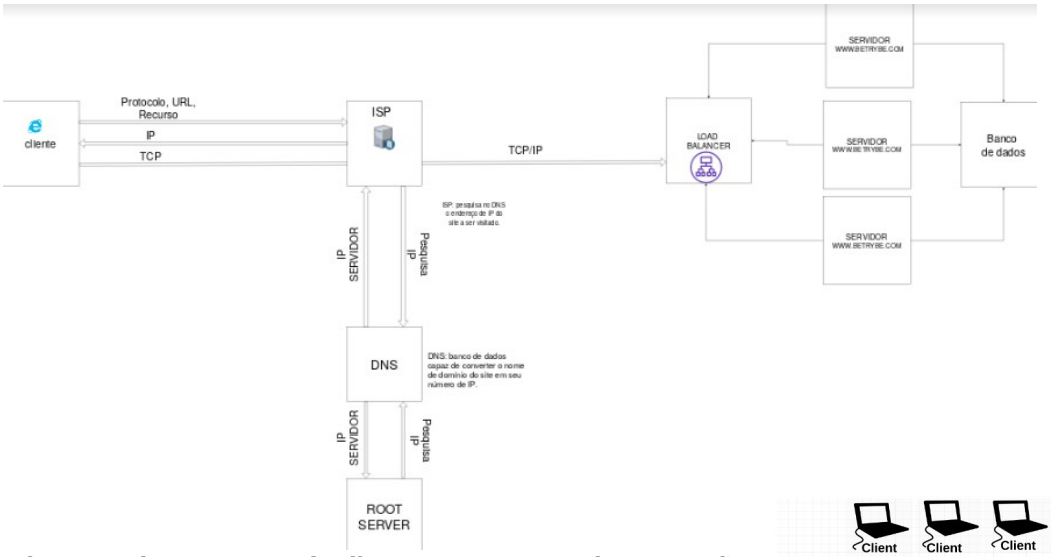
Examplo de fluxo: para atualizar sua branch 'form' com a 'master' e depois integrar o trabalho na master:

2) Internet

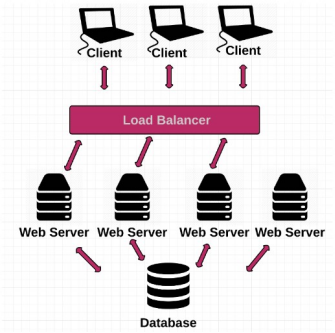
Software gerencia dados / **Hardware** cabos fisicos.
Modem transforma info em sinal / **Roteador** cria rotas (destinos via IP s).
Internet é infraestrutura / **Web** é o serviço cosntruido para essa infraestrutura.
Client pega interaçoes do user e transforma em request, browser, front-end / **Server** pega e responde requests, tipos diversos: web, database, file ou app servers.

Definitions IP address, ISP provenciador, DNS, DN, TCP/IP, Port number, Host, HTTP/https (default port 80/443), URL… : https://medium.com/free-code-camp/how-the-web-works-a-primer-for-newcomers-to-web-development-or-anyone-really-b4584e63585c#.8nc8nkx2u .

Journey from code to webpage:



(tabalho de alumno, pode ter erro, nos detalhes apenas para visualizar o geral)



3 BASIC ELEMENTS of app configuration:
CLIENT what the user interacts with
SERVER what listens to requests coming in from the client
DATABASE basement of web architecture.

Referência para entender modelo client-server: https://medium.com/free-code-camp/how-the-web-works-part-ii-client-server-model-the-structure-of-a-web-application-735b4b6d76e3#.e6tmj8112 .

To get SCALE:
LOAD BALANCER (balanceador de carge) routes client requests across several servers, distribute traffic via algorithms (round robin or least connections)
SERVICE another server interacting with servers, allows to break the server into multiple services packets with different functions
CONTENT DELIVERY NETWORK to make it faster, large distributed system of proxy servers.