COTI INFORMÁTICA
ESCOLA DE NERDS

GitHub

Java WebDeveloper – Formação FullStack

Professor Sergio Mendes

Aula 02 (11/01/23)

COTI INFORMÁTICA
ESCOLA DE NERDS

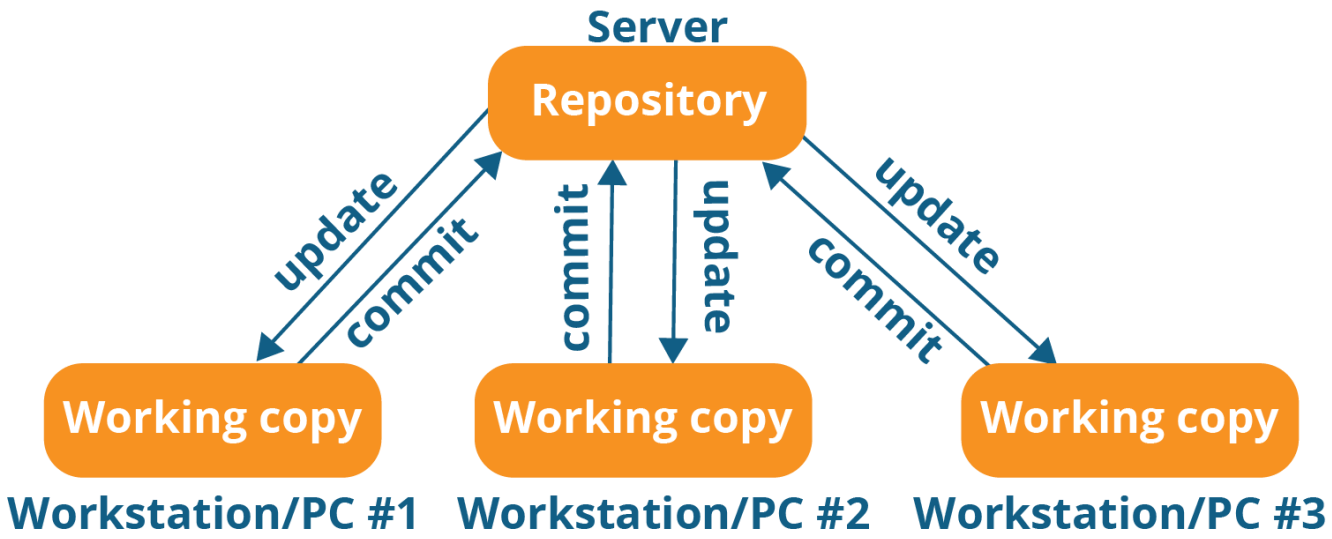
O que é o **GitHub**?

O GitHub é considerado é uma ferramenta essencial para engenheiros de software, com uma popularidade sem igual. Atualmente, ele acomoda mais de 25 milhões de usuários. Isso significa que há um número considerável de profissionais que estão procurando o GitHub para melhorar o fluxo de trabalho e a colaboração.

Em suma, o GitHub é um serviço baseado em nuvem que hospeda um sistema de controle de versão (VCS) chamado Git. Ele permite que os desenvolvedores colaborem e façam mudanças em projetos compartilhados enquanto mantêm um registro detalhado do seu progresso.

O que é um sistema de controle de versão?

Sempre que desenvolvedores criam um novo projeto eles continuam criando atualizações no código base. Mesmo depois de o projeto ser lançado é comum a atualização de versões, correção de bugs, adição de novas ferramentas, etc. O sistema de controle de versão ajuda a acompanhar as mudanças feitas no código base. E mais, ele também registra quem efetuou a mudança e permite a restauração do código removido ou modificado.

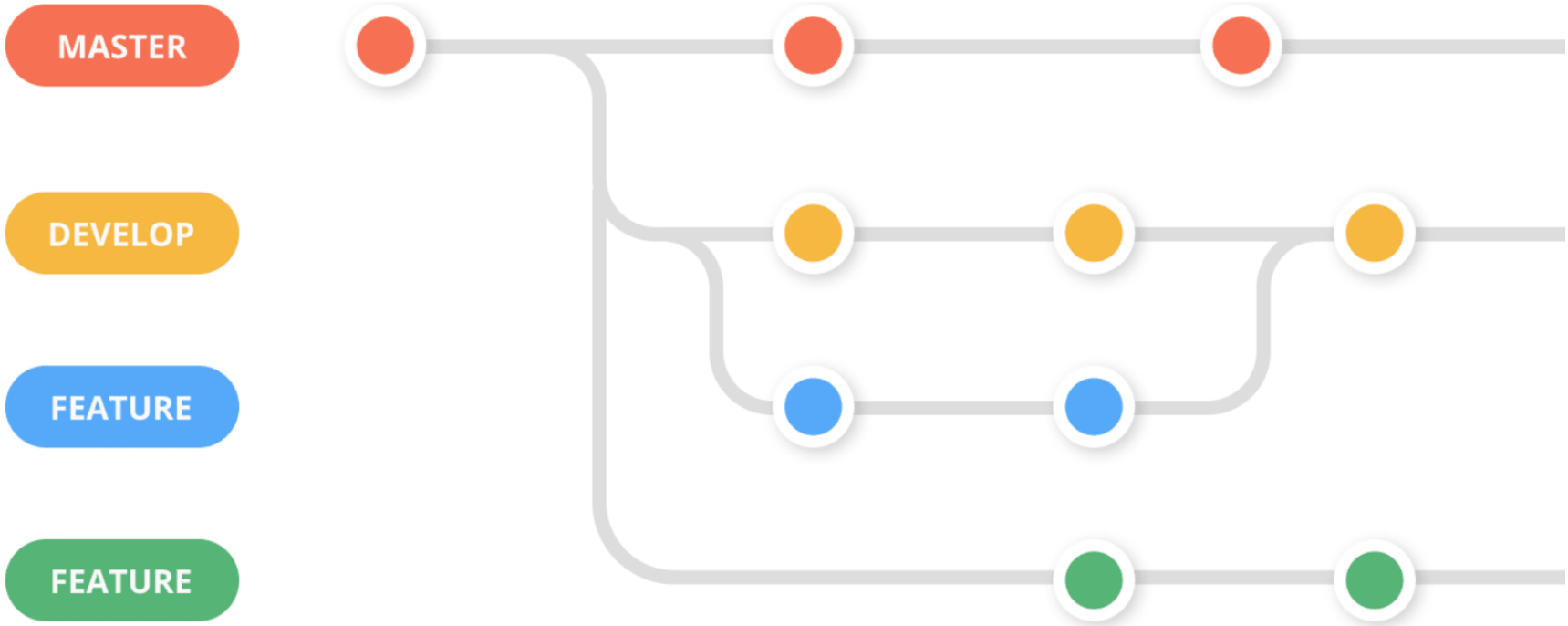


O Que é Git?

Agora que você sabe o que é GitHub, temos que entender que o Git é o coração do GitHub. Git é um sistema de controle de versão desenvolvido por Linus Torvalds (o criador do Linux).
Isso significa que qualquer desenvolvedor numa equipe pode gerenciar o código-fonte e seu histórico de mudanças usando ferramentas de linha de comandos de Git – desde que tenha sido concedido o acesso para isso, é claro.

Diferentemente dos sistemas de controle de versão centralizados, o Git oferece ramificações de recursos (ou *feature branches*). Isso significa que cada engenheiro de software na equipe pode separar uma ramificação de recursos que oferece um repositório local isolado para promover mudanças nos códigos.

Feature branches não afetam a ramificação principal, que é onde o código original do projeto está localizado. Uma vez que as mudanças tenham sido feitas e o código atualizado está pronto, a ramificação pode ser misturada (num processo de *merge*) com o *master branch*. É assim que as mudanças no projeto se tornam efetivas.



Por Que o GitHub é Tão Popular?

O GitHub hospeda mais de 100 milhões de repositórios, com a maior parte deles sendo projetos de código aberto.

Essa estatística mostra que o GitHub está entre os clientes de Git GUI mais populares, e também porque é usado por vários profissionais e grandes empresas, como a Hostinger.

Isso acontece porque o GitHub é um projeto de gestão baseado em nuvem e uma plataforma de organização que incorpora os recursos de controle de versão do Git. Isso significa que todos os usuários do GitHub podem acompanhar e gerenciar as mudanças feitas para o código-fonte em tempo real, enquanto têm acesso a todos os outros recursos do Git disponíveis no mesmo lugar.

Além disso, a interface de usuário do GitHub é mais amigável do que a do Git, fazendo com que seja mais acessível para pessoas que possuem pouco ou nenhum conhecimento técnico. Isso significa mais membros de equipe podem ser incluídos no progresso e na gestão do projeto, fazendo com que o processo seja mais tranquilo.

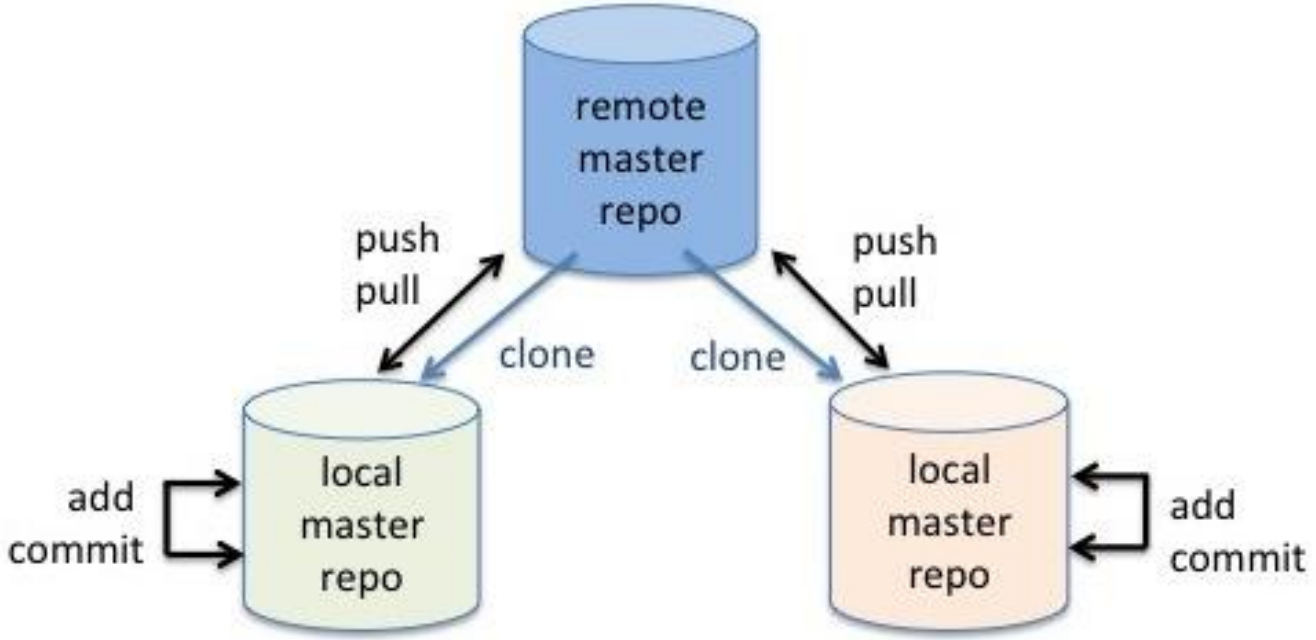
Como Começar a Usar o GitHub

Você pode experimentar o GitHub com a sua equipe de graça. Existe um plano básico disponível que inclui repositórios e colaboradores ilimitados, mas oferece apenas 500 MB de espaço de armazenamento.

Para aproveitar melhor os muitos recursos do GitHub, você pode escolher um dos planos pagos que eles oferecem.

1. Crie um Repositório no GitHub

Repositório, ou *repo*, é um diretório onde os arquivos do seu projeto ficam armazenados. Ele pode ficar em um depósito do GitHub ou em seu computador. Você pode armazenar códigos, imagens, áudios, ou qualquer outra coisa relacionada ao projeto no diretório.

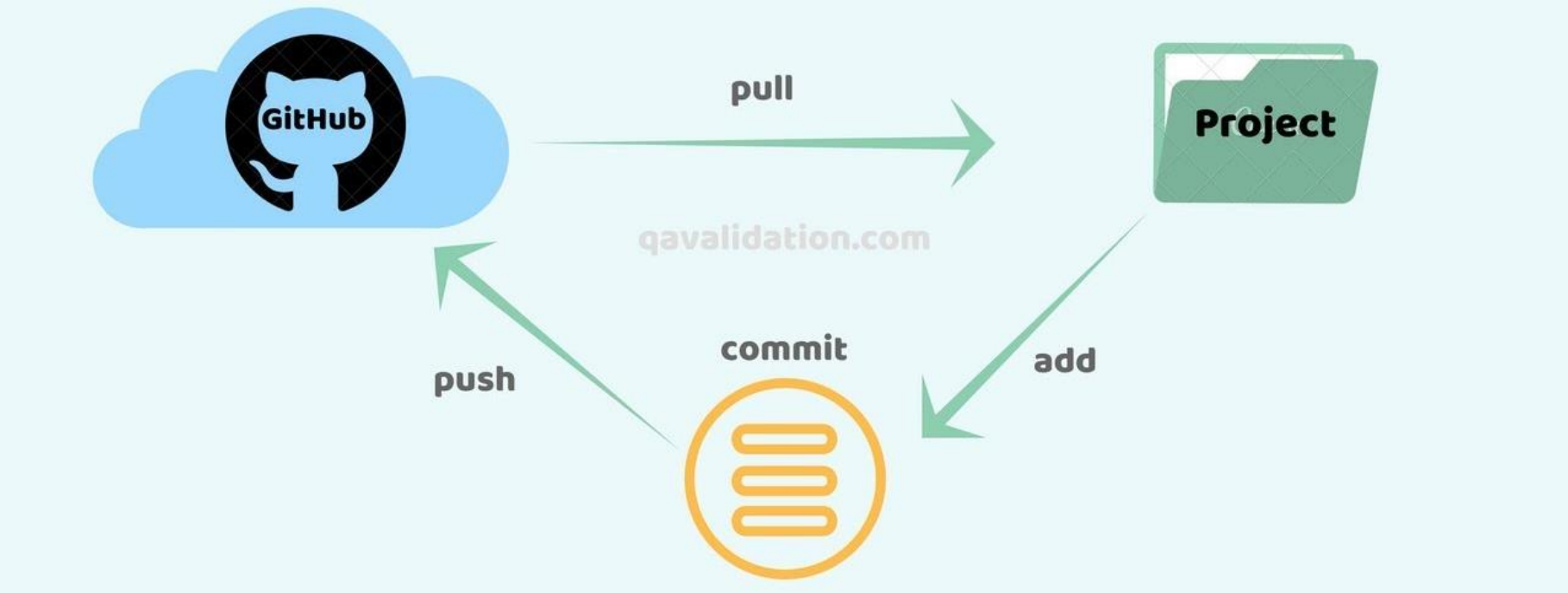


2. Crie Branches no GitHub

Ao criar *branches*, ou ramificações, você gera versões diferentes de um repositório. Quando você modifica o projeto nas *branches* de recursos, um desenvolvedor pode ver como isso vai afetar o projeto principal na hora que tudo for integrado.

3. Entenda Como Funcionam Commits no GitHub

Os Commits é como as mudanças salvas no GitHub são chamadas. Cada vez que você muda o arquivo do *branch* de recurso, você terá que executar um **Commit** para mantê-lo.



4. Crie Pull Requests no GitHub

Para propor as mudanças que você acabou de fazer para outros desenvolvedores trabalhando no mesmo projeto, você deve criar um **pull request**.

São eles que fazem ser tão fácil de trabalhar junto em projetos, já que eles são a principal ferramenta de colaboração no GitHub.

Pull Requests permitem que você veja as diferenças entre o projeto original e o seu *branch* de recurso. É assim que você pede para os seus pares revisá-las. Se os outros desenvolvedores aprovarem as modificações, eles podem executar um **merge pull request** (solicitação de mesclagem). Isso irá aplicar as mudanças para o projeto principal.

GitHub Não é Apenas Para Desenvolvedores

O GitHub é uma ótima plataforma que mudou o método de trabalho de desenvolvedores. Mas qualquer pessoa que deseja gerenciar seu projeto com eficiência e trabalhar com outros colaboradores também pode usar o GitHub.

Se sua equipe trabalha em um projeto que realiza atualizações constantes e você quer acompanhar como as mudanças são feitas, então o GitHub é uma ótima opção para você.

Existem outras alternativas como o **GitLab** e **BitBucket**, mas o GitHub deve ser levado em consideração.

O GitHub é seguro para grandes projetos?

Se você chegou até aqui, provavelmente está considerando usar o GitHub, certo? Então, só falta explicarmos porque a plataforma é segura para qualquer projeto que você possua, principalmente dada a grandiosidade e quantidade de usuários da ferramenta.

A segurança é um fator muito importante para o GitHub, que se preocupa com a privacidade dos dados e dá a opção para aqueles que preferirem, assinarem um pacote para manter os dados em servidores próprios.

Os criadores da plataforma estão constantemente melhorando as barreiras securitárias, além de seguirem as leis de proteção de informação da Europa, que muito se assemelham a LGPD aqui do Brasil. Portanto, é perfeitamente seguro usar o GitHub para grandes projetos.

Create a Repository

From scratch -- Create a new local repository

```
$ git init [project name]
```

Download from an existing repository

```
$ git clone my_url
```

Observe your Repository

List new or modified files not yet committed

```
$ git status
```

Show the changes to files not yet staged

```
$ git diff
```

Show the changes to staged files

```
$ git diff --cached
```

Show all staged and unstaged file changes

```
$ git diff HEAD
```

Show the changes between two commit ids

```
$ git diff commit1 commit2
```

List the change dates and authors for a file

```
$ git blame [file]
```

Show the file changes for a commit id and/or file

```
$ git show [commit]:[file]
```

Show full change history

```
$ git log
```

Show change history for file/directory including diffs

```
$ git log -p [file/directory]
```

Working with Branches

List all local branches

```
$ git branch
```

List all branches, local and remote

```
$ git branch -av
```

Switch to a branch, my_branch, and update working directory

```
$ git checkout my_branch
```

Create a new branch called new_branch

```
$ git branch new_branch
```

Delete the branch called my_branch

```
$ git branch -d my_branch
```

Merge branch_a into branch_b

```
$ git checkout branch_b
$ git merge branch_a
```

Tag the current commit

```
$ git tag my_tag
```

Make a change

Stages the file, ready for commit

```
$ git add [file]
```

Stage all changed files, ready for commit

```
$ git add .
```

Commit all staged files to versioned history

```
$ git commit -m "commit message"
```

Commit all your tracked files to versioned history

```
$git commit -am "commit message"
```

Unstages file, keeping the file changes

```
$ git reset [file]
```

Revert everything to the last commit

```
$ git reset --hard
```

Synchronize

Get the latest changes from origin (no merge)

```
$ git fetch
```

Fetch the latest changes from origin and merge

```
$ git pull
```

Fetch the latest changes from origin and rebase

```
$ git pull --rebase
```

Push local changes to the origin

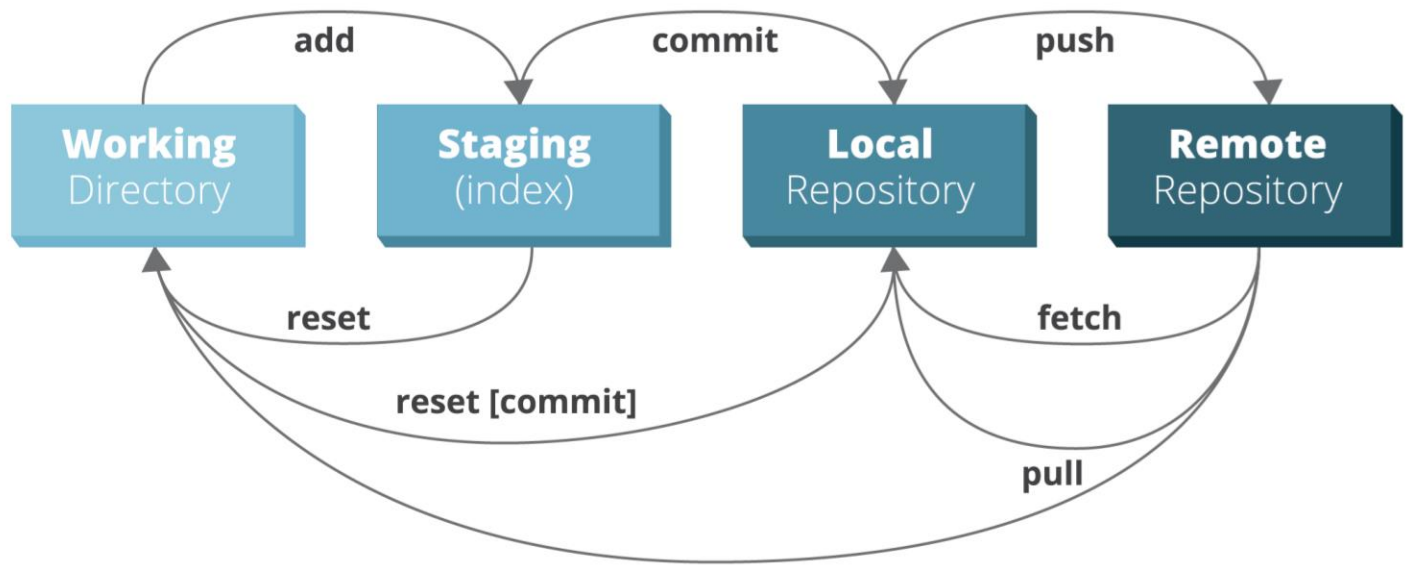
```
$ git push
```

Finally!

When in doubt, use git help

```
$ git command --help
```

Or visit **https://training.github.com/** for official GitHub training.



Git Cheat Sheet

Setup

Set the name and email that will be attached to your commits and tags

```
$ git config --global user.name "Danny Adams"
$ git config --global user.email "my-email@gmail.com"
```

Start a Project

Create a local repo (omit <directory> to initialise the current directory as a git repo

```
$ git init <directory>
```

Download a remote repo

```
$ git clone <url>
```

Make a Change

Add a file to staging

```
$ git add <file>
```

Stage all files

```
$ git add .
```

Commit all staged files to git

```
$ git commit -m "commit message"
```

Add all changes made to tracked files & commit

```
$ git commit -am "commit message"
```

Basic Concepts

main: default development branch
origin: default upstream repo
HEAD: current branch
HEAD^: parent of HEAD
HEAD~4: great-great grandparent of HEAD

By @DoableDanny

Branches

List all local branches. Add -r flag to show all remote branches. -a flag for all branches.

```
$ git branch
```

Create a new branch

```
$ git branch <new-branch>
```

Switch to a branch & update the working directory

```
$ git checkout <branch>
```

Create a new branch and switch to it

```
$ git checkout -b <new-branch>
```

Delete a merged branch

```
$ git branch -d <branch>
```

Delete a branch, whether merged or not

```
$ git branch -D <branch>
```

Add a tag to current commit (often used for new version releases)

```
$ git tag <tag-name>
```

Merging

Merge branch a into branch b. Add --no-ff option for no-fast-forward merge

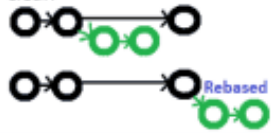

```
$ git checkout b
$ git merge a
```

Merge & squash all commits into one new commit

```
$ git merge --squash a
```

Rebasing

Rebase feature branch onto main (to incorporate new changes made to main). Prevents unnecessary merge commits into feature, keeping history clean




```
$ git checkout feature
$ git rebase main
```

Interatively clean up a branches commits before rebasing onto main

```
$ git rebase -i main
```

Interatively rebase the last 3 commits on current branch

```
$ git rebase -i Head~3
```

Undoing Things

Move (&/or rename) a file & stage move

```
$ git mv <existing_path> <new_path>
```

Remove a file from working directory & staging area, then stage the removal

```
$ git rm <file>
```

Remove from staging area only

```
$ git rm --cached <file>
```

View a previous commit (READ only)

```
$ git checkout <commit_ID>
```

Create a new commit, reverting the changes from a specified commit

```
$ git revert <commit_ID>
```

Go back to a previous commit & delete all commits ahead of it (revert is safer). Add --hard flag to also delete workspace changes (BE VERY CAREFUL)

```
$ git reset <commit_ID>
```

Review your Repo

List new or modified files not yet committed

```
$ git status
```

List commit history, with respective IDs

```
$ git log --oneline
```

Show changes to unstaged files. For changes to staged files, add --cached option

```
$ git diff
```

Show changes between two commits

```
$ git diff commit1_ID commit2_ID
```

Stashing

Store modified & staged changes. To include untracked files, add -u flag. For untracked & ignored files, add -a flag.

```
$ git stash
```

As above, but add a comment.

```
$ git stash save "comment"
```

Partial stash. Stash just a single file, a collection of files, or individual changes from within files

```
$ git stash -p
```

List all stashes

```
$ git stash list
```

Re-apply the stash without deleting it

```
$ git stash apply
```

Re-apply the stash at index 2, then delete it from the stash list. Omit stash@{n} to pop the most recent stash.

```
$ git stash pop stash@{2}
```

Show the diff summary of stash 1. Pass the -p flag to see the full diff.

```
$ git stash show stash@{1}
```

Delete stash at index 1. Omit stash@{n} to delete last stash made

```
$ git stash drop stash@{1}
```

Delete all stashes

```
$ git stash clear
```

Synchronizing

Add a remote repo

```
$ git remote add <alias> <url>
```

View all remote connections. Add -v flag to view urls.

```
$ git remote
```

Remove a connection

```
$ git remote remove <alias>
```

Rename a connection

```
$ git remote rename <old> <new>
```

Fetch all branches from remote repo (no merge)

```
$ git fetch <alias>
```

Fetch a specific branch

```
$ git fetch <alias> <branch>
```

Fetch the remote repo's copy of the current branch, then merge

```
$ git pull
```

Move (rebase) your local changes onto the top of new changes made to the remote repo (for clean, linear history)

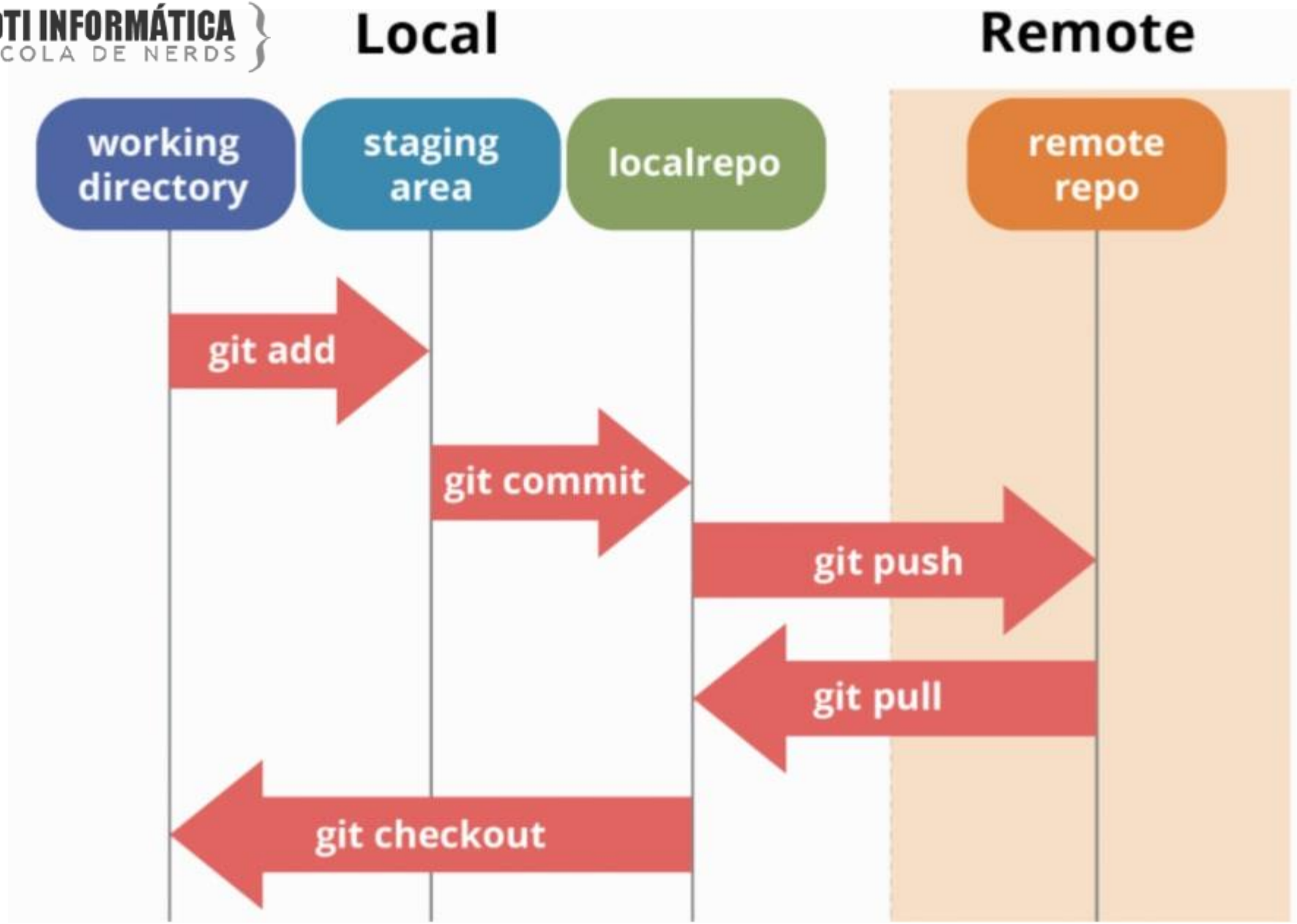
```
$ git pull --rebase <alias>
```

Upload local content to remote repo

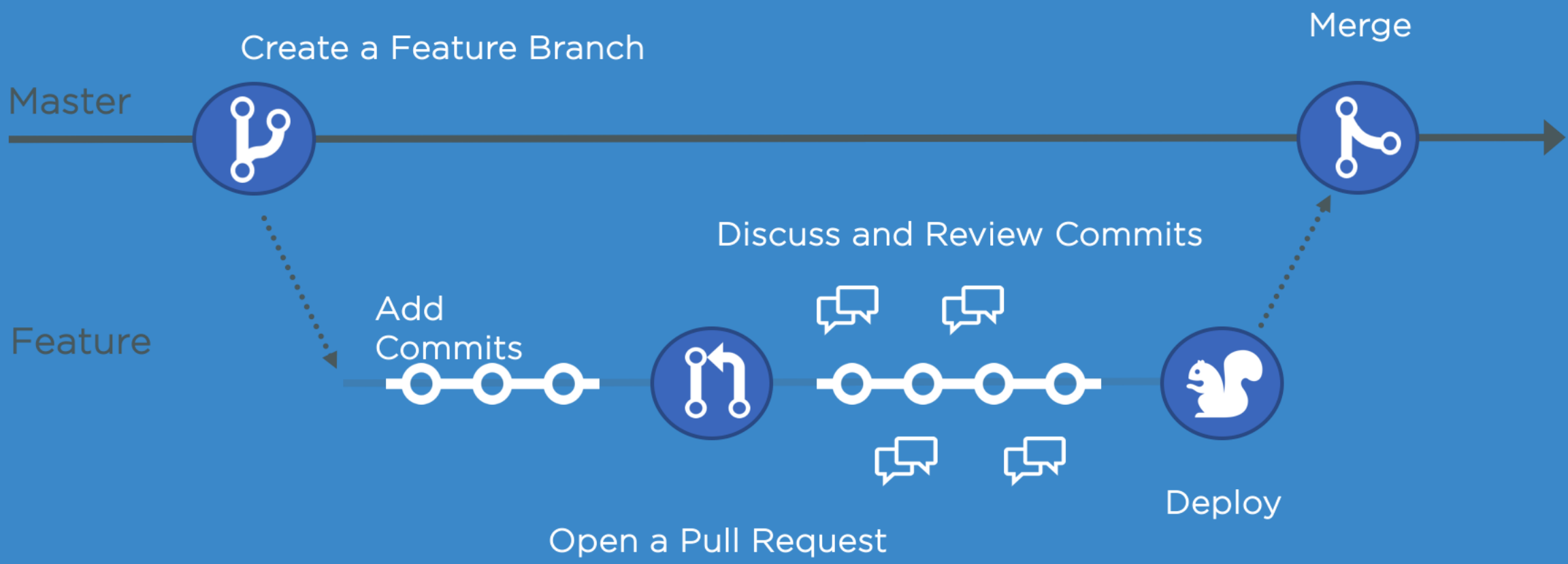
```
$ git push <alias>
```

Upload to a branch (can then pull request)

```
$ git push <alias> <branch>
```


Local
Remote
working directory
staging area
localrepo
remote repo
git add
git commit
git push
git pull
git checkout

GitHub Flow


Create a Feature Branch
Merge
Master
Discuss and Review Commits
Add
Commits
Feature
Deploy
Open a Pull Request

{ COTI INFORMÁTICA
ESCOLA DE NERDS }