

# 1º REUNIÃO DO PROJETO

GPP/MDS - GRUPO 08

# APRESENTAÇÃO

TEMPO PREVISTO: 10 MIN

# APRESENTAÇÃO DO TIME

## **- Integrantes de GPP**

**Luís Resende**

**Marco Willian**

**Matheus Godinho**

**Pedro Salles**

**Thiago Bernardes**

## **- Integrantes de MDS**

**André Rodrigues**

**Guilherme Ribeiro**

**Weyler Almeida**

# **DINÂMICA**

TEMPO PREVISTO: 5 MIN

# COMUNICAÇÃO

TEMPO PREVISTO: 10 MIN

# **FERRAMENTAS DE COMUNICAÇÃO**

**Google Drive**

**Telegram**

**Hangouts**



# FORMULÁRIOS

TEMPO PREVISTO: 10 MIN

# **FORMULÁRIOS PARA PREENCHIMENTO**

**Contatos**

**Disponibilidade do Time**

**Quadro de Conhecimentos**

**Definir agenda de reuniões**





# **PRINCÍPIOS DO RUP**

TEMPO PREVISTO: 15 MIN

# **PRINCÍPIOS DO RUP**

**Concepção**

**Elaboração**

**Construção**

**Transição**

**Principios Papeis**

**Principais Artefatos**




# DOJO DE GIT

TEMPO PREVISTO: 30 MIN


# O QUE É VERSIONAMENTO?

Um sistema de controle de versão na função prática da Ciência da Computação e da Engenharia de Software, é um software com a finalidade de gerenciar diferentes versões no desenvolvimento de um documento qualquer. Esses sistemas são comumente utilizados no desenvolvimento de software para controlar as diferentes versões — histórico e desenvolvimento — dos códigos-fontes e também da documentação.



# O QUE É VERSIONAMENTO?

Um sistema de controle de versão na função prática da Ciência da Computação e da Engenharia de Software, é um software com a finalidade de gerenciar diferentes versões no desenvolvimento de um documento qualquer. Esses sistemas são comumente utilizados no desenvolvimento de software para controlar as diferentes versões — histórico e desenvolvimento — dos códigos-fontes e também da documentação.



# CRIAR UM REPOSITÓRIO

O primeiro passo para trabalhar com Git é criar um repositório.

Para isso, basta rodar o comando:

```
git init NOME_DO_REPOSITORIO
```

*Exemplo:*

`git init games` cria a pasta chamada games, que é um repositório Git.

# STATUS DOS ARQUIVOS

Para verificar o status do git , você usa o comando:

`git status`

*Exemplo:*

`git status`: mostra o status atual do git e dos arquivos.

# STATUS DOS ARQUIVOS

- Do ponto de vista do Git, existem quatro estados no qual um determinado arquivo pode estar.

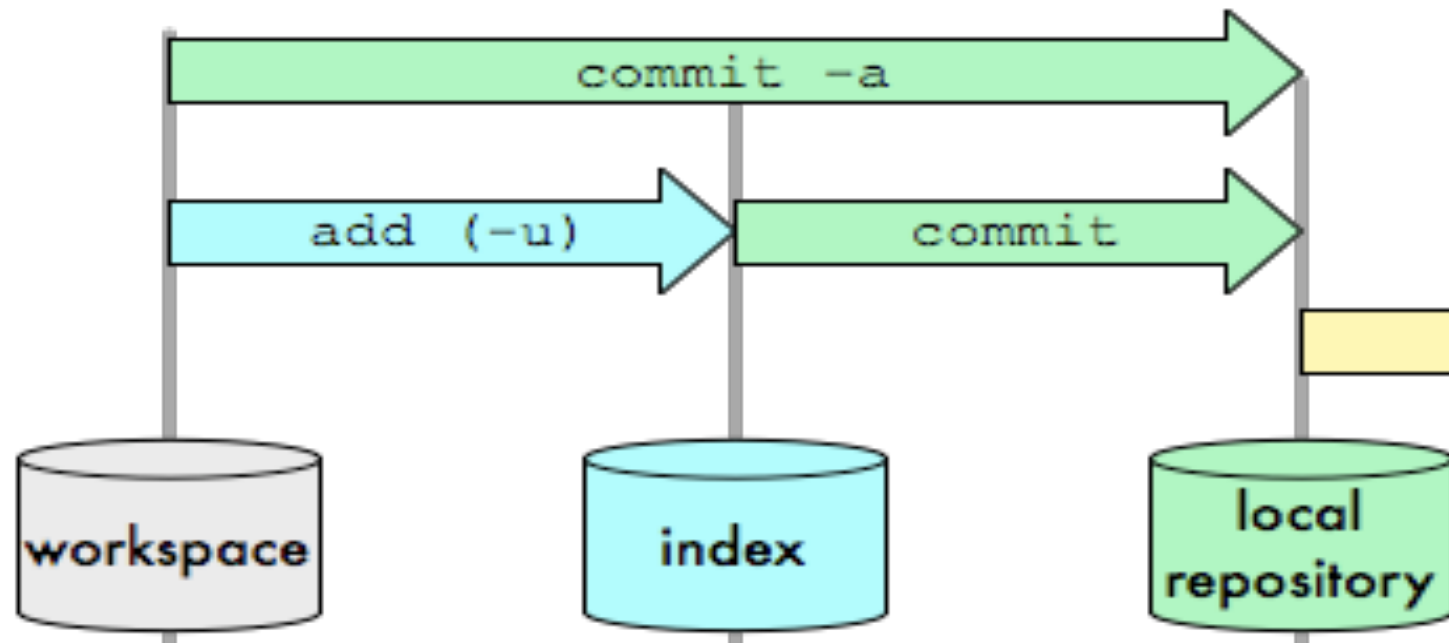
**Committed:** O arquivo está sob controle de versão e não apresenta nenhuma modificação.

**Commit candidate:** Caso você crie um conjunto de modificações (commit), esses serão os arquivos incluídos.

**Modified:** Arquivos que foram modificados.

**Untracked:** Arquivos novos, que ainda não estão no controle de versão.





# ADICIONAR UM ARQUIVO

Para mudar um arquivo de *untracked* ou *modified* para

*Commit candidate*, você usa o comando:

**git add CAMINHO**

*Exemplo:*

**git add .** mudaria o estado de todos os arquivos a partir do diretório atual.

**git add arquivo.txt** mudaria o estado do arquivo arquivo.txt

# FAZER COMMITS

Agora que se sabe como selecionar as modificações que quer incluir em um pacote de modificações (um *commit*), chegou a hora de aprender como gerar esse pacote.

O comando utilizado para criar um commit em git é **git commit**. Com esse comando, todos os arquivos que estiverem marcados como Commit candidate serão incluídos no commit.

*Exemplo:*

**git commit -m "Primeiro Commit"** cria um commit com todos os *Commit candidate* e a mensagem Primeiro Commit.

# CRIAR BRANCHES

Um *branch* nada mais é que uma lista de commits. É possível criar, apagar e renomear *branches*.

Para criar você usa o comando: `git branch NOME_DO_BRANCH`.

Para apagar: `git branch -D NOME_DO_BRANCH`.

Para renomear você usa: `git branch -m NOME_ANTIGO NOME_NOVO`.

*Exemplo:*

`git branch work` cria o branch work

`git branch -D work` apaga o branch work

`git branch -m work ModestoTeam` renomeia o branch work para ModestoTeam

# MUDAR DE BRANCH ATIVO

Agora que já sabe criar novos branches, podemos começar a trabalhar nesses branches criados.

Para você mudar o *branch* ativo (branch onde são colocados os seus commits), você usa o comando:

**git checkout**

*Exemplo:*

**git checkout work** muda para o branch work que você tenha criado previamente.

# MERGE DE BRANCHES

Como sempre trabalhamos com *branches*, uma operação muito útil é juntar código de *branches* diferentes.

No Git, uma das formas de se fazer isso é o Merge e o comando para fazer o merge é:

```
git merge BRANCH_DE_ORIGEM
```

*Exemplo:*

`git merge work` moverá todos os commits (acima da base) da *branchwork* para a *branch* atual.

# REBASE DE BRANCHES

O **Rebase** funciona de forma muito semelhante ao **Merge**. A diferença é a ordem em que os commits são aplicados.

No **Merge**, os commits do outro *branch* são aplicados por cima dos commits do *branch* atual.

No **Rebase**, os seus commits (acima da base) são temporariamente apagados, o branch atual fica exatamente igual ao outro branch e seus commits são aplicados um a um no *branch* atual.

Para fazer o Rebase, você usa o comando:

**git rebase BRANCH\_DE\_ORIGEM**

*Exemplo:*

**git rebase master** moverá todos os commits (acima da base) da *branch* master para a *branch* atual, mas abaixo dos seus commits acima da base.

# ENVIAR COMMITS PARA BRANCHES REMOTOS (PUSH)

Para enviar os commits do seu *branch* atual para o servidor remoto você usa o comando:

**git push REPO\_REMOTO BRANCH.**

*Exemplo:*

**git push origin master** envia os commits do seu branch atual para o branch master do repositório origin



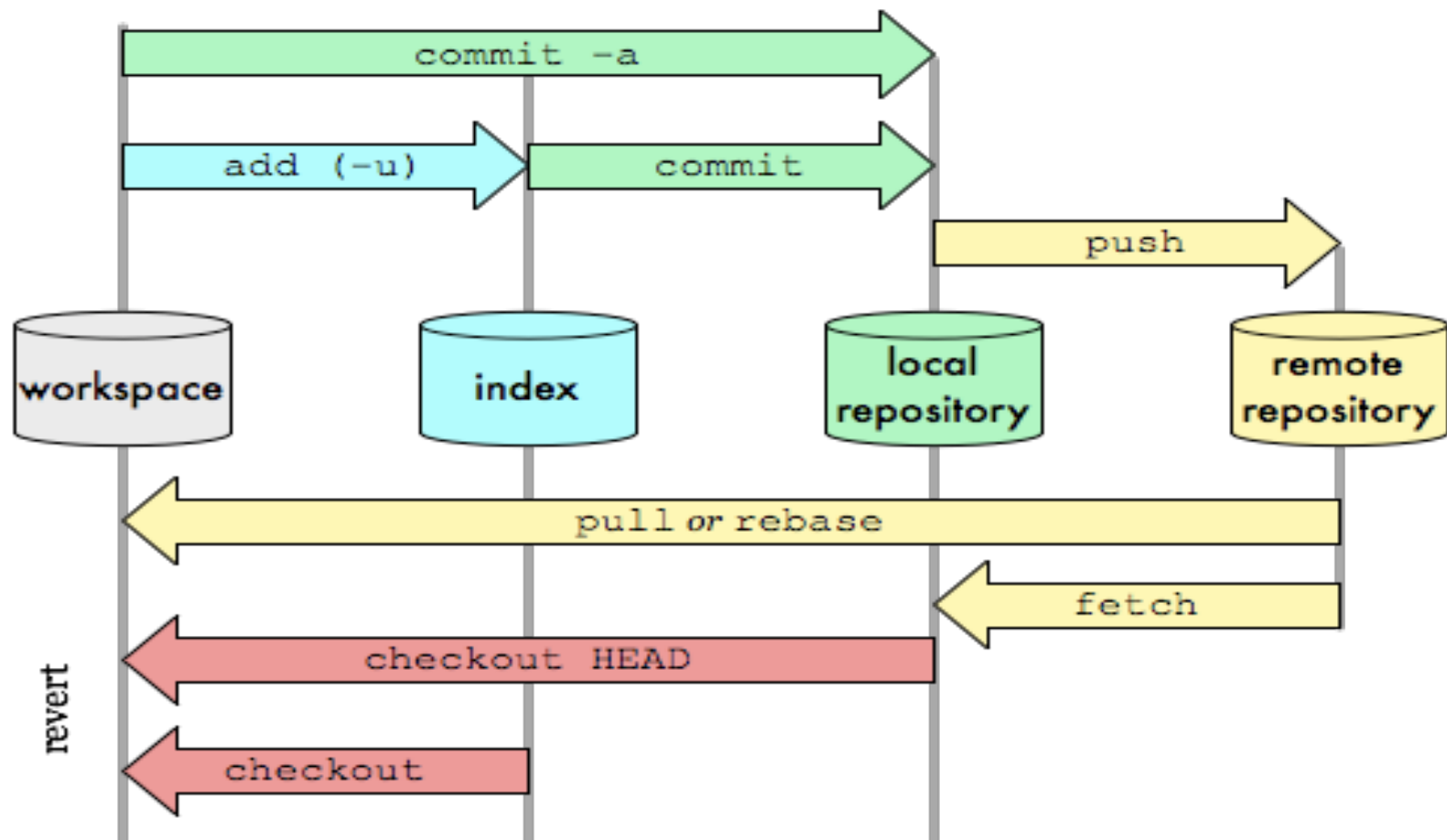
# PEGAR COMMITS DE BRANCHES REMOTOS (PULL)

Para trazer commits de um repositório remoto você usa o comando:

`git pull REPO_REMOTO BRANCH.`

*Exemplo:*

`git pull origin master` pega os commits do branch master do servidor remoto origin e aplica ao seu branch atual.



# GIT WORKFLOW

Uma das principais mudanças que acontecem quando você passa a usar Git é que o seu fluxo de trabalho muda.

Um fluxo de trabalho muito comum com **GIT** funciona da seguinte forma:

1. *Checkout* para o seu branch de trabalho;
2. Programar e fazer **vários pequenos commits**;
3. *Pegar as modificações* que outros programadores tenham feito;
4. *Rebase* no seu branch de trabalho (para manter o histórico linear);
5. *Enviar as modificações* para o servidor remoto.