

# Sumário

<b>1</b>	<b>Guia de termos</b>	<b>2</b>
<b>2</b>	<b>Fluxo comum de uso de git</b>	<b>3</b>
2.1	Primeira vez que for usar . . . . .	3
2.2	Lista de comandos . . . . .	3
2.3	Importante saber sobre os comandos . . . . .	4
<b>3</b>	<b>Dicas básicas</b>	<b>6</b>
3.1	Terminal e editor de texto . . . . .	6
3.2	DEU MERDA!!! . . . . .	7
<b>4</b>	<b>Github</b>	<b>7</b>
4.1	Conta com email da Unicamp . . . . .	7
4.2	Instruções gerais do Github . . . . .	8
<b>5</b>	<b>Links úteis</b>	<b>9</b>

# 1 Guia de termos

1. **repositório:** um diretório controlado por git. Pode ser local (um diretório no seu computador) ou remoto (github, gitlab, etc). branch: é uma ramificação do seu projeto, você pode criar ramos, trocar o branch que você está controlando no momento ou fundir (dar merge em) dois branches. Olhe os itens 7, 8, 9 e 10 do exemplo de fluxo abaixo.
2. **master:** é o ramo (branch) mais importante do projeto, imagine ele como o tronco da árvore e os outros branches são os ramos que saem dele. Idealmente, tudo que está no master são versões do código que já foram testadas em outros branches e depois fundidas com o master ou antes de terem sido adicionadas ao git em um commit.
3. **HEAD:** é o último commit do branch que você está controlando no momento. Imagine que você acabou de chegar em um checkpoint de um jogo, e tem alguma coisa que indica que aquele foi o último save que você deu caso você tente carregar o jogo. O HEAD é parecido, ele GERALMENTE mostra qual foi o último commit que você deu no branch atual. Se você mudar pra outro branch, ele atualiza o HEAD pro último commit desse branch, se você der um commit o HEAD vai pra esse novo commit.
4. **origin:** é o repositório remoto que seu repositório local usa de referência. Basicamente, é na origin que você publica o código que você alterou e adicionou ao git localmente e busca códigos feitos por outras pessoas para modificar localmente.

## 2 Fluxo comum de uso de git

### 2.1 Primeira vez que for usar

O git usa seu nome e email para manter informações de quem está dando commits em algum projeto. Por isso, a primeira vez que você for usar git em uma conta nova do seu computador (ou na sua conta no IC, por exemplo), você vai precisar dizer quem é você e qual seu email. Apenas utilize os seguintes comandos:

```
$ git config --global user.email <seuemail@exemplo.com>
```

```
$ git config --global user.name <seu lindo nome>
```

### 2.2 Lista de comandos

1. Criamos um repositório git

```
$ git init
```

2. Adicionamos um repositório remoto (origin)

```
$ git remote add origin <url do repositório remoto>
```

3. Copiamos o que existe na origin

```
$ git pull origin <nome do branch, normalmente usamos o master>
```

4. Adicionamos arquivos ao git

```
$ git add <nome dos arquivos ou diretórios>
```

5. Utilizamos commit para efetivar as últimas mudanças (como salvar um checkpoint)

*\$ git commit -m " Sua mensagem de commit aqui "*

6. Mantemos uma visão do estado atual do repositório

*\$ git status*

7. Criamos eventuais branches (ramos)

*\$ git branch <nome do branch>*

8. Alteramos o branch que estamos modificando (como dar load em um checkpoint).

*\$ git checkout <nome do branch>*

9. Mantemos uma visão do estado da nossa árvore de commits e branches

*\$ git log*

10. Efetuamos merges dos branches com nosso master

*\$ git checkout <branch que você quer manter, normalmente o master>*

*\$ git merge <nome do branch que você está dando merge>*

11. Atualizamos nossa origin com as mudanças

*\$ git push origin master*

## **2.3 Importante saber sobre os comandos**

- Os itens 6 e 10 podem (e devem) ser executados o tempo todo para verificar se o repositório está do jeito que você acredita que esteja
- Os itens da seção 2.2 listam um fluxo razoavelmente completo do uso de git, nem sempre passamos por todas essas etapas, principalmente em projetos pequenos como um lab de mc202 =)

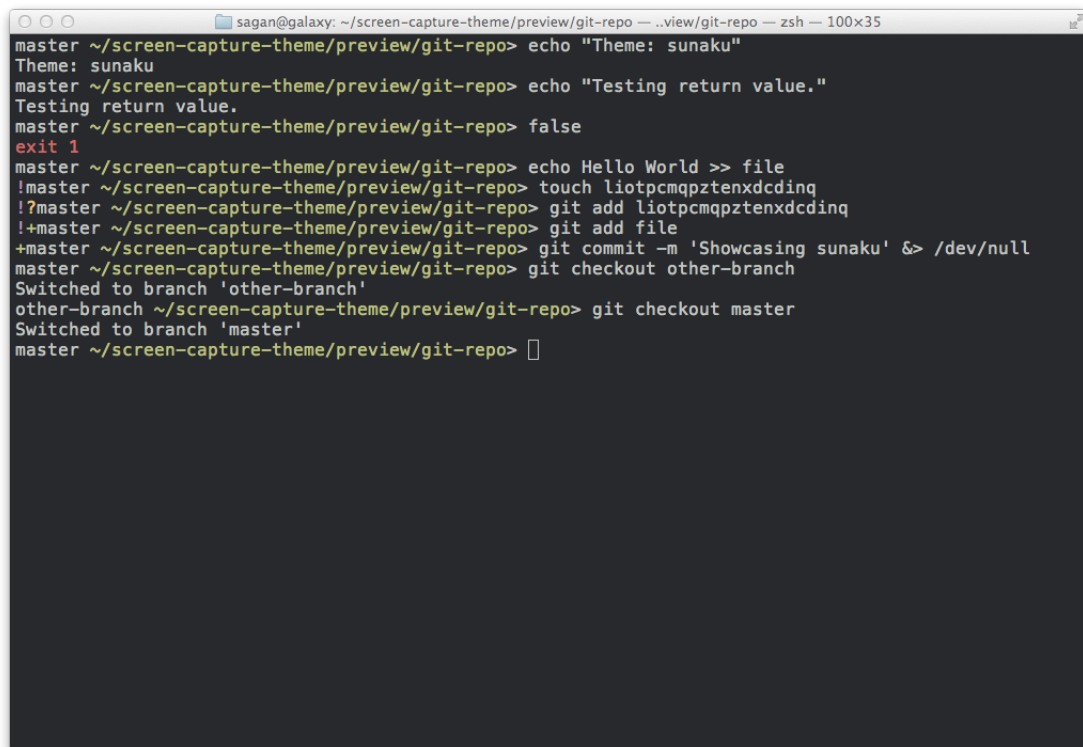
- No item 12 podemos colocar outros branches além do master na nossa origin, isso é comum quando nossos branches são grandes e temos mais de uma pessoa trabalhando neles
- **IMPORTANTE** - No item 11 podem ocorrer erros de conflito (imaginem que no master um arquivo tem uma linha X e no branch o mesmo arquivo não tem essa linha, qual das versões o git deve utilizar?). Podemos corrigir os conflitos na mão ou utilizar alguma flag para o git decidir qual versão usar (esse link explica bem como lidar com conflitos: <https://www.git-tower.com/learn/git/faq/solve-merge-conflicts>)
- **IMPORTANTE** - No item 8 dos exemplos anteriores, também podemos dar checkout para um commit passado (imagine que você descobriu um problema no último commit). Porém, essa não é a melhor prática de git, o ideal é só dar commit quando você está confiante de que as alterações até aquele momento estão corretas, ou pelo menos fazer essas gambiarras em branches que não sejam o master. **MANTENHA O MASTER O MAIS ORGANIZADO POSSÍVEL**, ele é o primeiro branch (se não o único) que a maioria das pessoas vai ver quando for olhar seu código

Quando mudamos de branch podemos modificar e adicionar os arquivos igualmente aos passos 4 e 5, mas agora essas modificações ocorrem no novo branch e não diretamente no master! (isso é bom, pq normalmente queremos que o master tenha apenas as mudanças que já foram testadas e efetivadas)

## 3 Dicas básicas

### 3.1 Terminal e editor de texto

- Configure seu terminal para ele mostrar o estado atual do seu repositório, por exemplo esse tema do Oh-My-Zsh:



```
sagan@galaxy: ~/screen-capture-theme/preview/git-repo — .view/git-repo — zsh — 100x35
master ~/screen-capture-theme/preview/git-repo> echo "Theme: sunaku"
Theme: sunaku
master ~/screen-capture-theme/preview/git-repo> echo "Testing return value."
Testing return value.
master ~/screen-capture-theme/preview/git-repo> false
exit 1
master ~/screen-capture-theme/preview/git-repo> echo Hello World >> file
!master ~/screen-capture-theme/preview/git-repo> touch liotpcmqpztenxdcdinq
!?master ~/screen-capture-theme/preview/git-repo> git add liotpcmqpztenxdcdinq
!+master ~/screen-capture-theme/preview/git-repo> git add file
+master ~/screen-capture-theme/preview/git-repo> git commit -m 'Showcasing sunaku' &> /dev/null
master ~/screen-capture-theme/preview/git-repo> git checkout other-branch
Switched to branch 'other-branch'
other-branch ~/screen-capture-theme/preview/git-repo> git checkout master
Switched to branch 'master'
master ~/screen-capture-theme/preview/git-repo> □
```

- Utilize um editor de texto com extensões para te ajudar a visualizar modificações no seu repositório enquanto você modifica o código, por exemplo Atom e vim

## 3.2 DEU MERDA!!!

Tá tudo bem, você pode sobrescrever as alterações locais do seu repositório:

```
$ git checkout -- <nome do arquivo que você quer resetar>
```

Isso vai resetar as modificações do arquivo para como ele estava no último commit do branch atual. É por isso que é MUITO IMPORTANTE manter o costume de dar commits organizados e com código minimamente testado.

Se deu muita merda mesmo e você quer resetar o repositório local e recomeçar de onde você parou na origin, você pode fazer isso:

```
$ git fetch origin
```

```
$ git reset --hard origin/master
```

Isso vai buscar a última versão do repositório remoto origin e depois mandar o git resetar tudo do repositório local para o estado atual da origin. Por isso que é AINDA MAIS IMPORTANTE ser cuidadoso com os pushes para a origin.

## 4 Github

### 4.1 Conta com email da Unicamp

O Github é lindo, mas em contas FREE ele não permite que você faça repositórios fechados, o que é péssimo para seus trabalhos individuais. Por sorte, é possível utilizar o email da unicamp para ter acesso aos repositórios privados de graça!

- Crie uma conta no Github com qualquer email

- Adicione seu gmail da dac (<primeira letra do seu nome><seu ra>@g.unicamp.br) à sua conta do Github em *Personal Settings > Emails > Add email address* (se você usou ele no cadastro pule essa etapa)
- Entre nesse link, ache a opção do Github e escolha a opção *Get direct access on the GitHub website*. Depois siga os passos do link.

## 4.2 Instruções gerais do Github

- Mantenha um README um LICENÇA nos repositórios do github, você nunca sabe se alguém está interessado no seu código
- Para criar um repositório remoto para um repositório local (ou seja, criar uma origin pro seu repositório local):
  1. Siga as instruções do próprio Github para criar um novo projeto, por exemplo: *repo-remoto*
  2. O Github vai fornecer alguns jeitos de adicionar coisas no repositório. Se você já possui um repositório local, basta copiar a url do repositório remoto que você acabou de criar (com o .git no final), vai ser algo como:  
*https://github.com/seuusuario/repo-remoto.git*
  3. Depois, siga o passo 2 da seção 2.2



## 5 Links úteis

- Tutorial básico bem explicado: [http://rogerdudler.github.io/git-guide/index.pt\\_BR.html](http://rogerdudler.github.io/git-guide/index.pt_BR.html)
- Introdução a repositórios do github: <https://guides.github.com/activities/hello-world/>
- Readme lindo do Chams: <https://github.com/mrtheduts/readmegit>