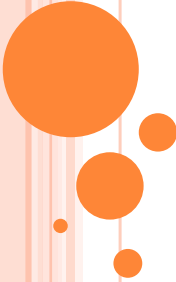



# INTRODUÇÃO AO GIT



Gibeon Aquino  
gibeon@dimap.ufrn.br

## O QUE É?

Git é um sistema de controle de versão distribuído e um sistema de gerenciamento de código fonte, com ênfase em velocidade



## COMO SURTIU?

- Tudo começou em Abril de 2005
- No ambiente que circunscreve o desenvolvimento do Kernel Linux
- Após muitos desenvolvedores perderem o acesso ao ambiente/ferramenta usado na época: Bitkeeper
- Linus Torvalds decidiu criar uma solução própria

## DEFINIÇÕES DO PROJETO

- Aplicação de um patch não deveria durar mais que 3 segundos
- Ter o Concurrent Versions System (CVS) como um bom exemplo do que NÃO FAZER
- Suportar um fluxo de trabalho distribuído como o BitKeeper
- Incluir mecanismos fortes contra corrompimento de arquivos

## EM CONSTANTE EVOLUÇÃO

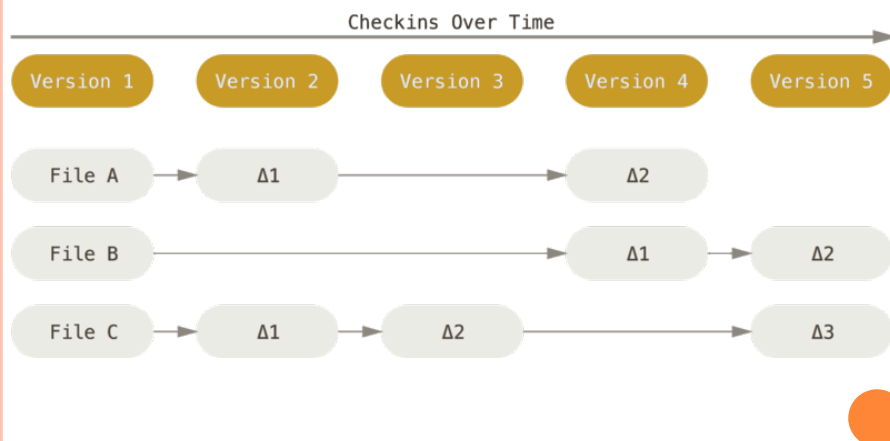
Version	Original release date <sup>[citation needed]</sup>	Latest (patch) version	Release date (of patch) <sup>[citation needed]</sup>
0.99	2005-07-11	0.99.9n	2005-12-15
1.0	2005-12-21	1.0.13	2006-01-27
1.1	2006-01-08	1.1.6	2006-01-30
1.2	2006-02-12	1.2.6	2006-04-08
1.3	2006-04-18	1.3.3	2006-05-16
1.4	2006-06-10	1.4.4.5	2008-07-16
1.5	2007-02-14	1.5.6.6	2008-12-17
1.6	2008-08-17	1.6.6.3	2010-12-15
1.7	2010-02-13	1.7.12.4	2012-10-17
1.8	2012-10-21	1.8.5.6	2014-12-17
1.9	2014-02-14	1.9.5	2014-12-17
2.0	2014-05-28	2.0.5	2014-12-17
2.1	2014-08-16	2.1.4	2014-12-17
2.2	2014-11-26	2.2.3	2015-09-04
2.3	2015-02-05	2.3.10	2015-09-29
2.4	2015-04-30	2.4.12	2017-05-05
2.5	2015-07-27	2.5.6	2017-05-05
2.6	2015-09-28	2.6.7	2017-05-05
2.7	2015-10-04	2.7.5	2017-05-05
2.8	2016-03-28	2.8.5	2017-05-05
2.9	2016-06-13	2.9.4	2017-05-05
2.10	2016-09-02	2.10.3	2017-05-05
2.11	2016-11-29	2.11.2	2017-05-05
2.12	2017-02-24	2.12.3	2017-05-05
2.13	2017-05-10	2.13.4	2017-08-01
2.14	2017-08-04	2.14.3	2017-10-24
2.15	2017-10-30	2.15.1	2017-11-28
2.16	2018-01-17	2.16.2	2018-02-15
2.17	2018-04-02	2.17.0	2018-04-02

Legend: ■ Old version ■ Older version, still supported ■ Latest version ■ Latest preview version

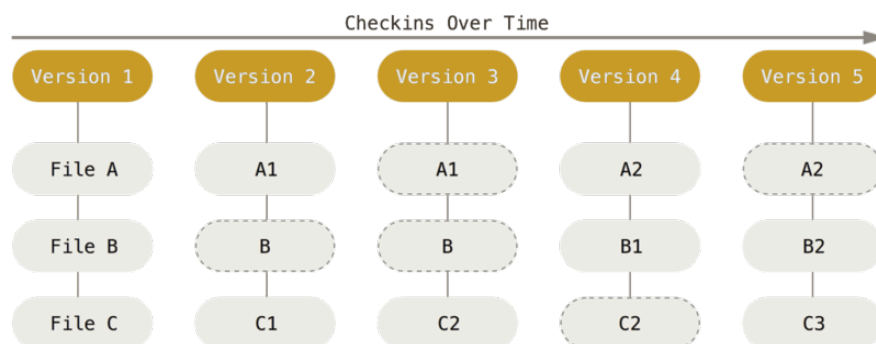
## CARACTERÍSTICA – ESTRUTURA INTERNA

- Vários Sistemas de controle de versão (ex. CVS, SVN) guardam os arquivos e o delta das mudanças realizadas
- MAS O Git é diferente!
  - Cada commit gera um snapshot do sistema de arquivos completos
  - Arquivos que não foram alterados não são copiados e sim referenciados


## OUTRAS FERRAMENTAS (CVS, SUBVERSION, PERFORCE, BAZAAR)




## GIT




### CARACTERÍSTICA – OPERAÇÕES LOCAIS

- Maior parte das operações são locais, sem necessidade de acesso a servidor remoto
  - Todo o histórico do projeto está armazenado localmente
- 


### CARACTERÍSTICA – INTEGRIDADE

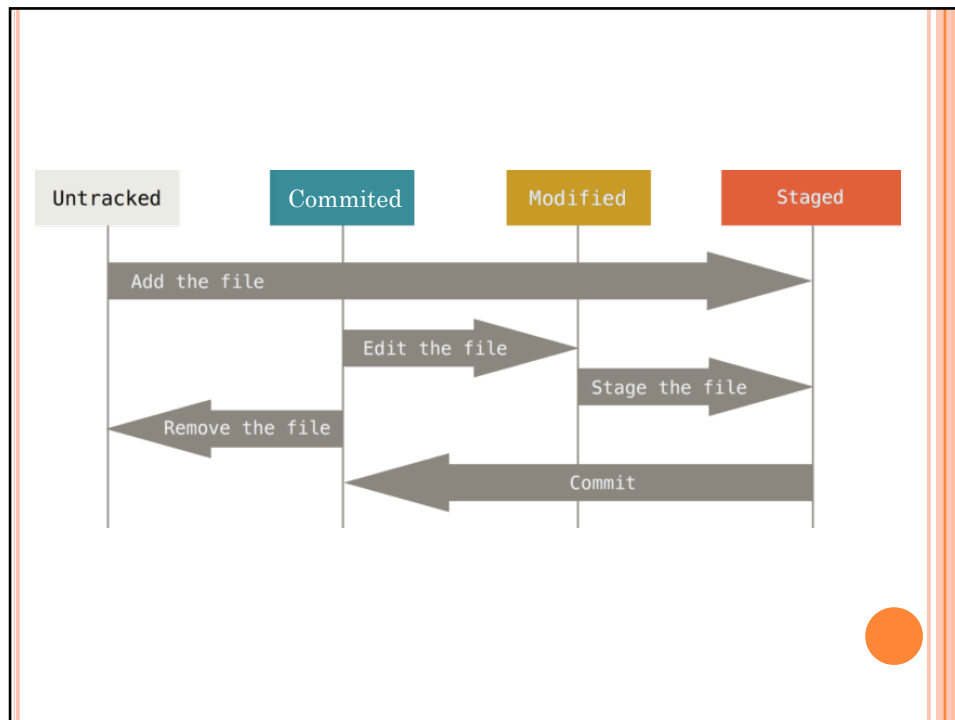
- Tudo é "Check-summed" antes de ser armazenado
  - O checksum é usado para referenciar os arquivos armazenados
  - Nenhuma informação pode ser perdida ou corrompida sem o Git detectar
- 

### CARACTERÍSTICA – APENAS ADICIONA DADOS

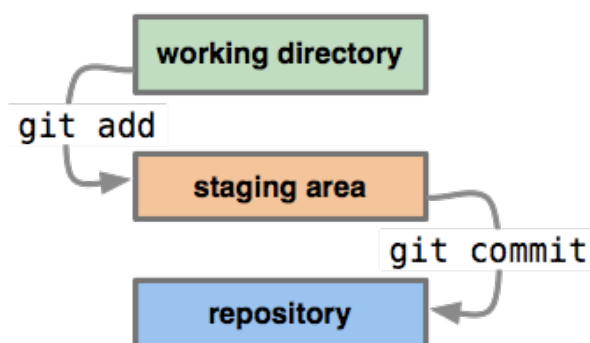
- Praticamente todas as ações no Git apenas adicionam dados ao seu banco de dados
  - Isso dá a liberdade de experimentar, sem medo de perder algo ou bagunçar as coisas
- 

### CARACTERÍSTICA – 3 ESTÁGIOS

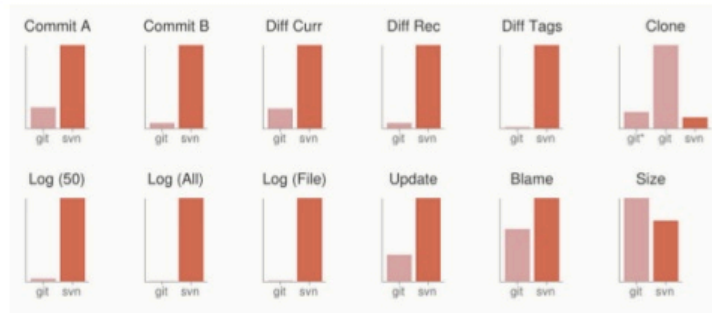
- Os arquivos podem residir em 3 estados:
  - **Committed** – O arquivo está seguro e armazenado localmente
  - **Modified** – O arquivo foi modificado e não foi commitado ainda
  - **Staged** – O arquivo modificado foi marcado em sua versão atual e será enviado no próximo commit
- 



## ÁREAS DO GIT



## Performance Benchmarks



source <https://git-scm.com>

## INSTALAÇÃO DO GIT

- Ver procedimento em <http://git-scm.com/downloads>



## COMANDOS GIT

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)

**clone** Clone a repository into a new directory  
**init** Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)

**add** Add file contents to the index  
**mv** Move or rename a file, a directory, or a symlink  
**reset** Reset current HEAD to the specified state  
**rm** Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)

**bisect** Use binary search to find the commit that introduced a bug  
**grep** Print lines matching a pattern  
**log** Show commit logs  
**show** Show various types of objects  
**status** Show the working tree status

grow, mark and tweak your common history

**branch** List, create, or delete branches  
**checkout** Switch branches or restore working tree files  
**commit** Record changes to the repository  
**diff** Show changes between commits, commit and working tree, etc  
**merge** Join two or more development histories together  
**rebase** Reapply commits on top of another base tip  
**tag** Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)

**fetch** Download objects and refs from another repository  
**pull** Fetch from and integrate with another repository or a local branch  
**push** Update remote refs along with associated objects

## UM POUCO MAIS...

available git commands in '/usr/local/git/libexec/git-core'

add	clone	fetch	interpret-trailers
add--interactive	column	fetch-pack	log
am	commit	filter-branch	ls-files
annotate	commit-tree	fmt-merge-msg	ls-remote
apply	config	for-each-ref	ls-tree
archimport	count-objects	format-patch	mailinfo
archive	credential	fack	mailsplit
bisect	credential-cache	fack-objects	merge
bisect--helper	credential-cache--daemon	gc	merge-base
blame	credential-store	get-tar-commit-id	merge-file
branch	cvsexportcommit	grep	merge-index
bundle	cvsmimport	gui	merge-octopus
cat-file	cvsserver	gui--askpass	merge-one-file
check-attr	daemon	hash-object	merge-ours
check-ignore	describe	help	merge-recursive
check-mailmap	diff	http-backend	merge-resolve
check-ref-format	diff-files	http-fetch	merge-subtree
checkout	diff-index	http-push	merge-tree
checkout-index	diff-tree	imap-send	mergetool
cherry	difftool	index-pack	mktag
cherry-pick	difftool--helper	init	mktree
citool	fast-export	init-db	mv
clean	fast-import	instaweb	name-rev
notes	remote-testsvn	submodule	
p4	repack	submodule--helper	
pack-objects	replace	subtree	
pack-redundant	request-pull	svn	
pack-refs	rerere	symbolic-ref	
patch-id	reset	tag	
prune	rev-list	unpack-file	
prune-packed	rev-parse	unpack-objects	
pull	revert	update-index	
push	rm	update-ref	
quiltimport	send-email	update-server-info	
read-tree	send-pack	upload-archive	
rebase	sh-i18n--envsubst	upload-pack	
rebase--helper	shell	var	
receive-pack	shortlog	verify-commit	
reflog	show	verify-pack	
remote	show-branch	verify-tag	
remote-ext	show-index	web--browse	
remote-fd	show-ref	whatchanged	
remote-ftp	stage	worktree	
remote-ftps	stash	write-tree	
remote-http	status		
remote-https	strippspace		

# GIT NA PRÁTICA...

## PASSO 1: CONFIGURAÇÃO DO AMBIENTE

- Definir nome e e-mail a ser usado na assinatura dos commits

```
$ git config --global user.name "Gibeon Aquino"  
$ git config --global user.email gibeon@dimap.ufrn.br
```

Todo commit no Git é assinado com esses dados

## PASSO 2: INICIANDO A ÁREA DE TRABALHO

- Criando um novo repositório **local** → `git init`

```
MacBook-Gibeon:helloworld3 gibeon$ git init
Initialized empty Git repository in /Users/gibeon/Google Drive/ACADEMICO/ENSINO/RESIDENCIA TI/GerenciamentoVersao/exemplos/helloworld3/.git/
MacBook-Gibeon:helloworld3 gibeon$
```

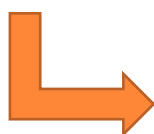
Cria um repositório Git vazio

```
MacBook-Gibeon:helloworld3 gibeon$ ls -al
total 0
drwxr-xr-x  3 gibeon  staff  102 May  8 21:16 .
drwxr-xr-x  6 gibeon  staff  204 May  8 21:20 ..
drwxr-xr-x 10 gibeon  staff  340 May  8 21:16 .git
```

## PASSO 2: INICIANDO A ÁREA DE TRABALHO

- Clonando um repositório existente → `git clone <url>`

```
MacBook-Gibeon:helloworld4 gibeon$ git clone https://github.com/gibeonaquino/helloworld
Cloning into 'helloworld'...
remote: Counting objects: 9, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 9 (delta 0), reused 9 (delta 0), pack-reused 0
Unpacking objects: 100% (9/9), done.
```



```
MacBook-Gibeon:helloworld gibeon$ ls -al
total 16
drwxr-xr-x  5 gibeon  staff  170 May  8 21:22 .
drwxr-xr-x  3 gibeon  staff  102 May  8 21:22 ..
drwxr-xr-x 13 gibeon  staff  442 May  8 21:22 .git
-rw-r--r--  1 gibeon  staff   13 May  8 21:22 README.md
-rw-r--r--  1 gibeon  staff   15 May  8 21:22 hello.txt
```

Cria uma cópia de um repositório remoto na sua máquina local

## EXERCICIO 1

- Criar ou usar uma conta do Github
- Criar um novo repositório através da interface do Github
  - Ativar a opção de criar repositório com um README
- Clonar o Repositório remoto
- Realizar as configurações globais de assinatura dos commits

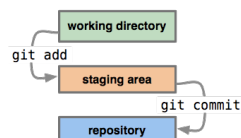
## PASSO 3: REALIZANDO COMMITS

- Um **Commit** no GIT apenas registra a mudança no repositório local
- Para cada commit o Git gera um **hash SHA-1** (string de 40 caracteres hexadecimais) como ID do commit

```
sha1(  
  commit message  
  commiter  
  commit date  
  author  
  author date  
  tree  
  parents  
)
```

## PASSO 3: REALIZANDO COMMITS

`git commit + git add`



```

[gibeon$ echo Conteudo Inicial >> file1.txt
[gibeon$ git add file1.txt
[gibeon$ git commit -m "Primeiro Commit"
[master e08e54a] Primeiro Commit
1 file changed, 2 insertions(+)
create mode 100644 file1.txt
  
```

## PASSO 3: REALIZANDO COMMITS

```

$ edit hello.c
$ git rm goodbye.c
$ git add hello.c
$ git commit
  
```

Qual o resultado?

```

$ edit hello.c
$ rm goodbye.c
$ git commit -a
  
```

Qual o resultado?

```

$ edit hello.c goodbye.c
$ git add goodbye.c
$ git add hello.c
$ git commit hello.c
  
```

Qual o resultado?

### PASSO 3: COMANDOS RELACIONADOS AO COMMIT

- `git log`
- `git status`
- `git reset`

### PASSO 3: COMANDOS RELACIONADOS AO COMMIT

#### **git log** - Apresenta o log dos commits

```
git log
git log --oneline
git log --stat
git log --after="2018-4-1"
git log --after="2018-4-1" --before = "2018-4-8"
git log --author="Gibeon"
git log master...hotfix01
```

## PASSO 3: COMANDOS RELACIONADOS AO COMMIT

### **git status** – Apresenta o estado atual do diretório de trabalho e da área de *staging*

```
gibeon$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
(use "git push" to publish your local commits)

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   file3.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   file1.txt

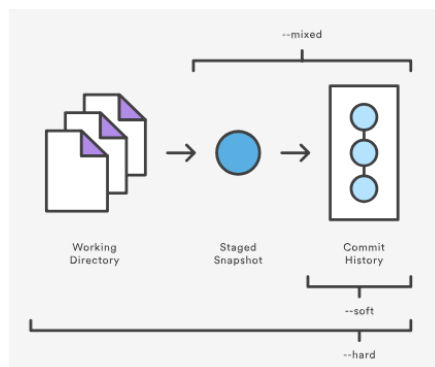
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    file2.txt
```

## PASSO 3: COMANDOS RELACIONADOS AO COMMIT

### **git reset** – comando para desfazer mudanças

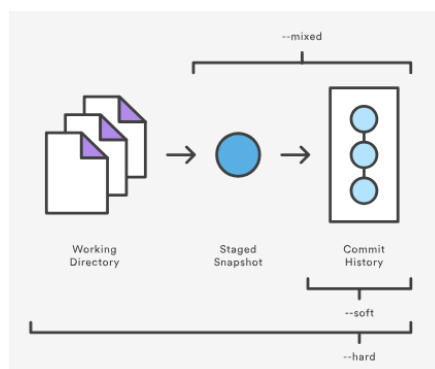
**3 opções principais:** argumentos `--soft`, `--mixed`, `--hard`



## PASSO 3: COMANDOS RELACIONADOS AO COMMIT

### git reset – comando para desfazer mudanças

**3 opções principais:** argumentos `--soft`, `--mixed`, `--hard`



## EXEMPLO 1 - RESET

```
# Edit both hello.py and main.py
# Stage everything in the current directory
git add .
# Realize that the changes in hello.py and main.py
# should be committed in different snapshots
# Unstage main.py
git reset main.py
# Commit only hello.py
git commit -m "Make some changes to hello.py"
# Commit main.py in a separate snapshot
git add main.py
git commit -m "Edit main.py"
```



## EXEMPLO 2 - RESET

```
# Create a new file called `foo.py` and add some code to it
# Commit it to the project history
git add foo.py
git commit -m "Start developing a crazy feature"
# Edit `foo.py` again and change some other tracked files
# Commit another snapshot
git commit -a -m "Continue my crazy feature"
# Decide to scrap the feature and remove the associated code
git reset --hard HEAD~2
```

## GIT RESET X GIT REVERT

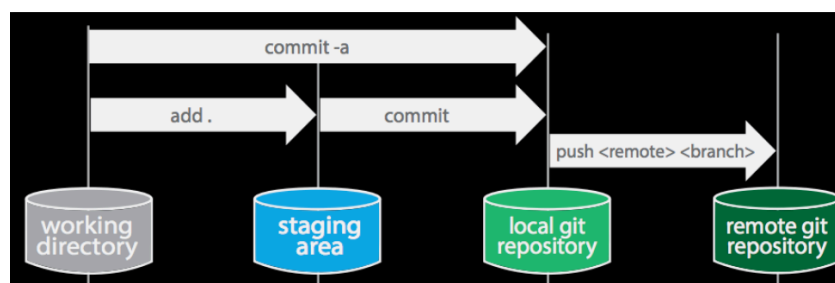
- **RESET** → Projetado para fazer mudanças locais no diretório de trabalho e área de *staging*
  - *Remove o rastro das mudanças*
- **REVERT** → Projetado para desfazer, de forma segura, *commits* públicos
  - Mantém o *rastro das mudanças* e usa um novo commit para desfazer as coisas

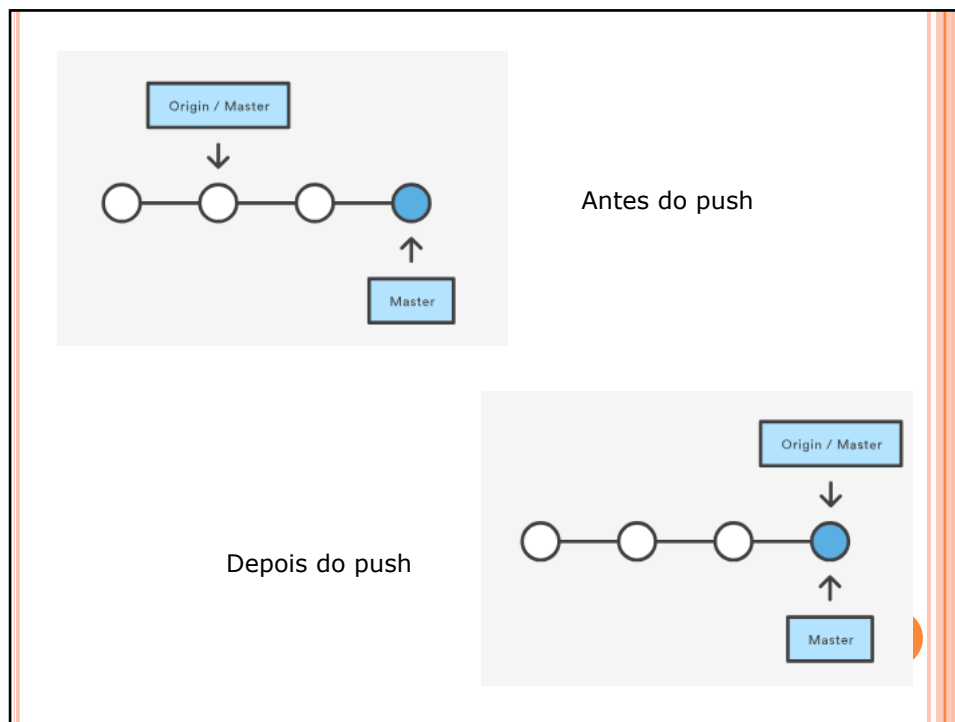
## EXERCICIO 2

- Crie em sua área de trabalho 2 arquivos file01.txt e file02.txt, adicione conteúdo a cada um deles.
  - Explore o comando `git status` antes de realizar commits
  - Adicione os arquivos à área de staging e explore o mesmo comando
  - Dê commit nas mudanças e explore o comando `git status` e `git log`
- Explore o comando `git reset` (COM CUIDADO)
  - Altere o arquivo file01.txt, adicione-o a área de staging e depois tente descartar esse *staging*

## PASSO 4: TRABALHANDO DE FORMA DISTRIBUÍDA

**git push** – Envia mudanças do repositório local para o remoto





## PASSO 4: TRABALHANDO DE FORMA DISTRIBUÍDA

```
git push <REMOTENAME> <BRANCHNAME>
```

Branch a ser enviada

```
git push origin master
```

Nome padrão do repositório clonado

```
git push <REMOTENAME> <LOCALBRANCHNAME>:<REMOTEBRANCHNAME>
```

```
git push origin bug003:hotfix_v2_1
```

```
git push
```

Envia todas as branches com nomes equivalentes no repositório remoto

## PASSO 4: TRABALHANDO DE FORMA DISTRIBUÍDA

**git pull** – Traz os commits do repositório remoto para o local (e *tenta fazer o merge com a branch local*)

```
git pull <remote>
```

## PASSO 4: TRABALHANDO DE FORMA DISTRIBUÍDA

**git fetch** – Traz os commits do repositório remoto para o local (não altera a branch local)

```
git fetch coworkers feature_branch  
fetching coworkers/feature_branch
```

*Uma maneira segura de revisar commits de outros, antes de integrá-los ao seu repositório*

## PASSO 4: TRABALHANDO DE FORMA DISTRIBUÍDA

**git fetch X git pull**



**git fetch + git merge**

## PASSO 4: TRABALHANDO DE FORMA DISTRIBUÍDA

**git remote** – Comando para criar, listar e remover conexões a repositórios remotos

```
gibeon$ git remote -v
origin  https://github.com/gibeonaquino/helloworld.git (fetch)
origin  https://github.com/gibeonaquino/helloworld.git (push)
```

```
gibeon$ git remote add repo2 https://github.com/gibeonaquino/helloworld2.git
```

*Por padrão, o Git já cria uma conexão quando o repositório é clonado*

## EXERCÍCIO 4

- Submeta as alterações do seu repositório local para o repositório remoto
- Crie um arquivo diretamente na interface do github e depois atualize seu repositório local
  - Explore as diferenças entre o pull e o fetch

## PASSO 5: TRABALHANDO COM BRANCHES

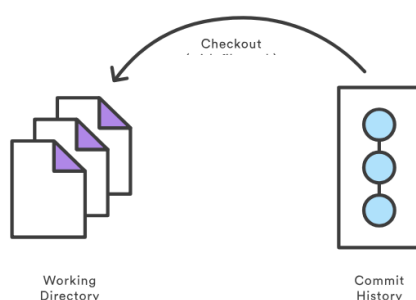
**git branch** – Lista, cria, renomeia ou remove branches

```
git branch # Lista as branches locais
git branch -a # Lista as branches locais e remotas
git branch <branch> # Cria uma branch
git branch -d <branch> # Remove a branch (seguro)
git branch -D <branch> # Remove a branch
git branch -m <branch> # Renomeia a branch
```

## PASSO 5: TRABALHANDO COM BRANCHES

**git checkout** – Permite chavear a área de trabalho entre diferentes versões do repositório

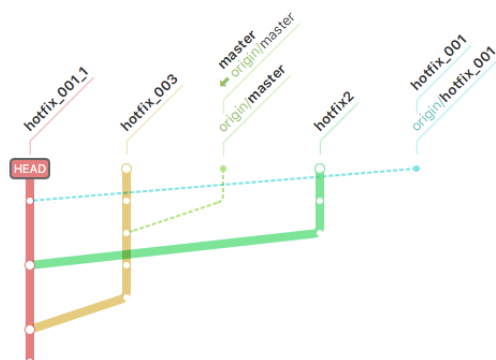
*Opera sobre arquivos, commits ou branches*



## PASSO 5: TRABALHANDO COM BRANCHES

**git checkout <branch>** # Inicia o trabalho na branch informada

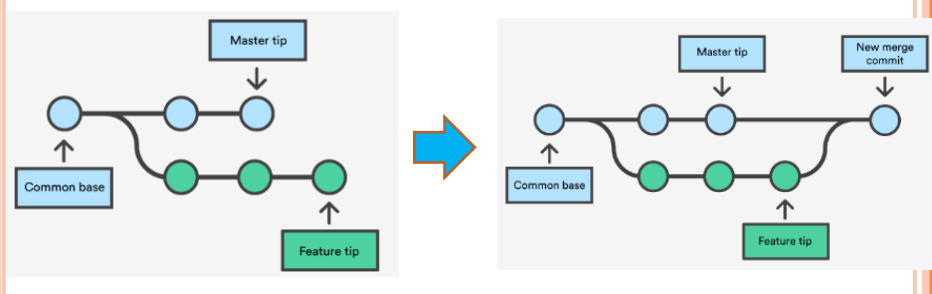
**git checkout -b <new-branch>** # Inicia o trabalho criando uma nova branch a partir da branch atual



## PASSO 5: TRABALHANDO COM BRANCHES

**git merge** – Integra duas branches em uma única

*Comando intrinsecamente relacionado aos resultados anteriores do **git checkout** e **git branch***



## ESTRATÉGIAS DO MERGE NO GIT

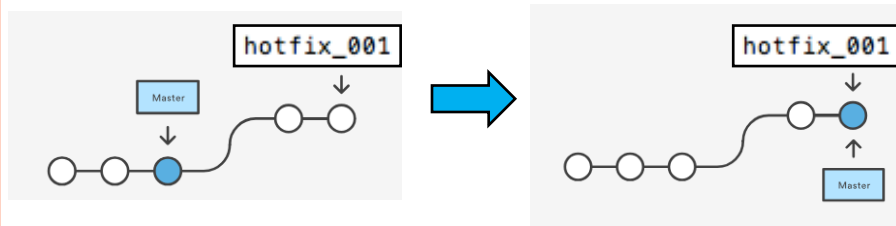
- Fast-Forwarding
- Three-Way Merge
- Conflicted Merge



## FAST-FORWARDING

- Usado quando há um caminho linear comum entre os branches participantes do merge

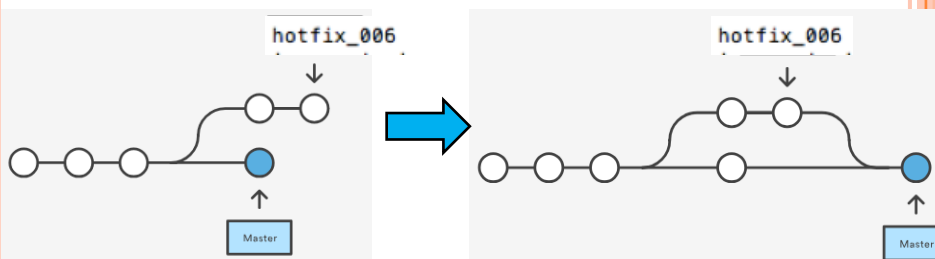
```
gibeon$ git checkout hotfix_001
Switched to branch 'hotfix_001'
gibeon$ git merge hotfix_001_1
Updating b0ea33e..28d03f9
Fast-forward
 hello.txt | 1 +
 1 file changed, 1 insertion(+)
```



## THREE-WAY MERGE

- Usado quando não há um caminho linear comum, mas há um ancestral em comum

```
gibeon$ git merge hotfix_006
Merge made by the 'recursive' strategy.
 file7.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 file7.txt
```



## CONFLICTED MERGE

- Conflito acontece quando os dois branches participantes do merge possuem alterações conflitantes nos arquivos alterados

```
gibeon$ git merge hotfix_004
Auto-merging file1.txt
CONFLICT (content): Merge conflict in file1.txt
Automatic merge failed; fix conflicts and then commit the result.
```

*Neste caso, o desenvolvedor precisa resolver o conflito manualmente*

## CONFLICTED MERGE

- Resolvendo o conflito...

```
gibeon$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

(1)
Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   file1.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

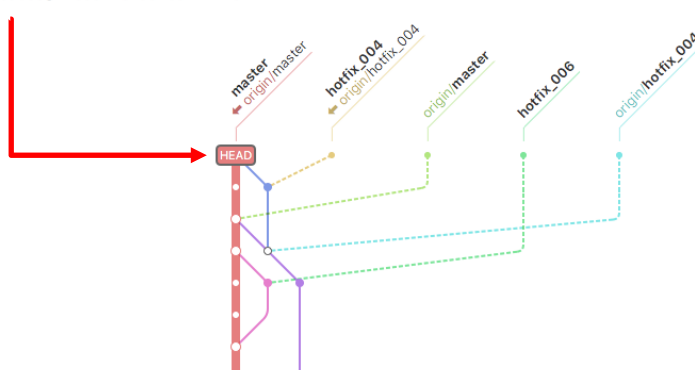
## CONFLICTED MERGE

### ○ Resolvendo o conflito...

**(2)** Edita o(s) arquivo(s) para resolver o conflito

**(3)** Dá o commit no arquivo com conflito resolvido...

```
gibeon$ git commit -am "Resolvendo conflito"
[master dbbad95] Resolvendo conflito
```



## EXERCÍCIO 5

- Crie uma branch hotfix01 e comece a trabalhar nela
  - Faça uma alteração no arquivo file02.txt
  - Dê um commit no arquivo
  - Envie a branch hotfix01 para o repositório remoto
- Integre as mudanças feitas em file02.txt à branch master
- Inspeccione o histórico de mudanças do seu repositório
- Submeta as alterações para o repositório remoto



## PASSO 6: SITUAÇÕES MAIS AVANÇADAS

○ `git commit --amend`

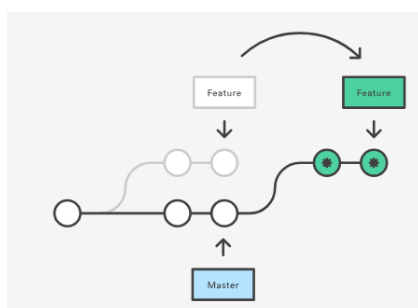
### Mecanismo útil para modificar o commit mais recente

- Alterar a mensagem do commit
- Adicionar novos arquivos ao commit
- Alterar conteúdo dos arquivos sem gerar um novo commit

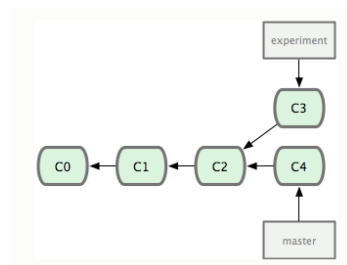
## PASSO 6: SITUAÇÕES MAIS AVANÇADAS

○ `git rebase`

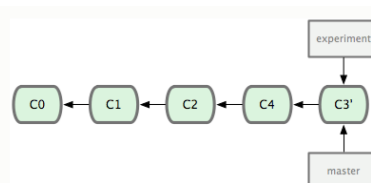
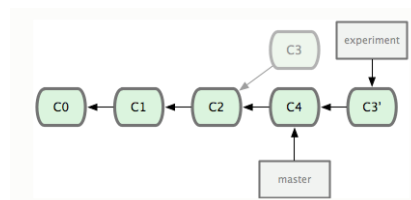
### Mecanismo útil para mover ou combinar uma sequencia de commits em um único commit base



## REBASING...



```
$ git checkout experiment
$ git rebase master
```



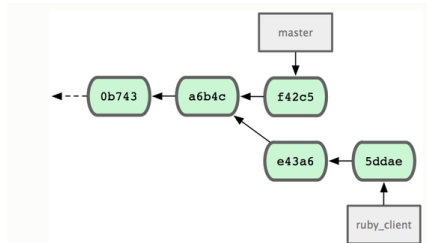
```
$ git checkout master
$ git merge experiment
```

## PASSO 6: SITUAÇÕES MAIS AVANÇADAS

- git cherry-pick

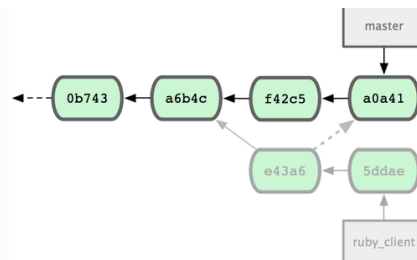
Aplica commits específicos a uma outra branch

## CHERRY PICKING...



```

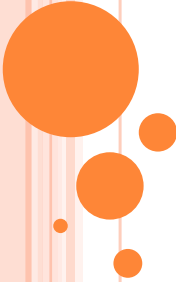
$ git cherry-pick e43a6fd3e94888d76779ad79fb568ed180e5fcd
Finished one cherry-pick.
[master]: created a0a41a9: "More friendly message when locking the index f
3 files changed, 17 insertions(+), 3 deletions(-)
  
```



## QUER SABER MAIS?

- <https://www.atlassian.com/git/tutorials>
- <https://git-scm.com/book/en/v2>

# INTRODUÇÃO AO GIT



Gibeon Aquino  
gibeon@dimap.ufrn.br