Versionamento de código com GIT

Prof. Msc. Aldisio Medeiros
aldisio.medeiros@lapisco.ifce.edu.br

Versionamento de código com Git

Versionamento de código com Git

- **O que é um controle de versão?**
  - Um sistema de controle de versão é um software (ou mais de um) que se encarrega de ***gerenciar as mudanças nos arquivos*** físicos. Esses arquivos podem ser de diferentes tipos, tais como **documentos, imagens e código fonte** de algum programa.
- Contar a história do seu projeto:

- Onde usar o controle de versão?
  - Desde sistema complexos até documentos de texto, como seu TCC / Monografia, aquela sua planilha de gastos no Excel, seu artigo, foto, etc

Versionamento de código com Git

- **Mas o que é gerenciar mudança?**
  - **Avisar** se teve mudança?
  - **Quem** fez tal mudança?
  - **Quando** esta foi feita?
  - Idealmente, **porque** esta alteração foi implementada?

- **Vantagens no uso de um controle de versão:**
  - **Segurança:** O acesso ao código é limitado.
  - **Versionamento:** Possibilita manter backups históricos.
  - **Rastreabilidade:** Mecanismos de identificação de falha por mudança.
  - **Colaboração:** Mecanismo que viabilizam o trabalho em equipe.
  - **Disponibilidade:** Código fonte na nuvem, acessível de qualquer lugar.

Versionamento de código com Git

- **Mas o que é gerenciar mudança?**

Revisão das 17h55min de 25 de outubro de 2019 (editar)
Douglasboavista (discussão | contribs)
m *(Foram revertidas as edições de 187.180.165.192 devido a vandalismo (usando Huggle) (3.2.0))*
(Etiquetas: Huggle, Reversão)
← Ver a alteração anterior

Revisão das 12h53min de 31 de outubro de 2019 (editar) (desfazer)
177.43.58.174 (discussão)
(→Principais vantagens)
(Etiqueta: Editor Visual)
Ver a alteração posterior →

Linha 1:

{{ver desambig|o sistema de controle de revisões utilizado na Wikipédia|Ajuda:Histórico de revisões}}

\- [[**Imagem**:Revision controlled project visualization-2010-24-02.svg|thumb|right|Exemplo da visualização do histórico de um projeto usando um sistema de controle de versões]]

\+ [[**Ficheiro**:Revision controlled project visualization-2010-24-02.svg|thumb|right|Exemplo da visualização do histórico de um projeto usando um sistema de controle de versões]]

Um '''sistema de controle de versões''' (ou "versionamento"), '''VCS''' (do inglês "[[:en:version control system|version control system]]") ou ainda '''SCM''' (do inglês "source code management") na função prática da [[Ciência da Computação]] e da [[Engenharia de Software]], é um "[[software]]" que tem a finalidade de gerenciar diferentes [[versão|versões]] no desenvolvimento de um documento qualquer. Esses sistemas são comumente utilizados no [[desenvolvimento de software|desenvolvimento de "software"]] para controlar as diferentes versões — histórico e desenvolvimento — dos [[código-fonte|códigos-fontes]] e também da [[documentação de software|documentação]].

Linha 17:

* '''Ramificação de projeto''': a maioria das implementações possibilita a divisão do projeto em várias linhas de desenvolvimento, que podem ser trabalhadas paralelamente, sem que uma interfira na outra.

* '''Segurança''': Cada software de controle de versão usa mecanismo para evitar qualquer tipo de invasão de agentes infecciosos nos arquivos. Além do mais, somente usuários com permissão poderão mexer no código.

\- * '''Rastreabilidade''': com a necessidade de sabermos o local, o estado e a qualidade de um arquivo; o controle de versão **trás** todos esses requisitos de forma que o usuário possa se embasar do arquivo que deseja utilizar.

\+ * '''Rastreabilidade''': com a necessidade de sabermos o local, o estado e a qualidade de um arquivo; o controle de versão **traz** todos esses requisitos de forma que o usuário possa se embasar do arquivo que deseja utilizar.

\- * '''Organização''': Com o software é **disponibilizado** interface visual **que pode** ser **visto todo** arquivos controlados, desde a origem até o projeto por completo.

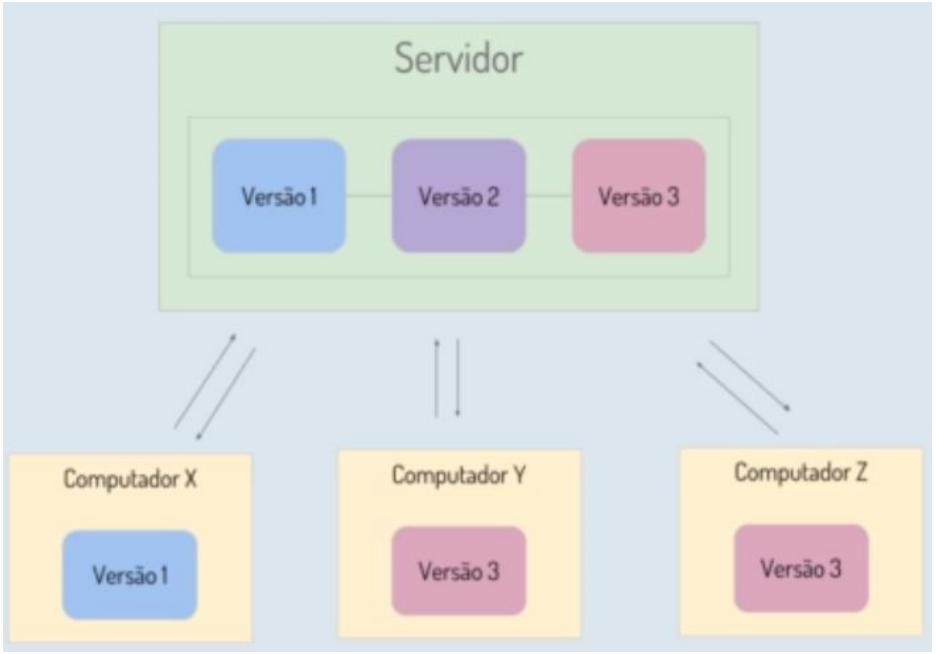
\+ * '''Organização''': Com o software é **disponibilizada** interface visual **onde podem** ser **vistos todos os** arquivos controlados, desde a origem até o projeto por completo.

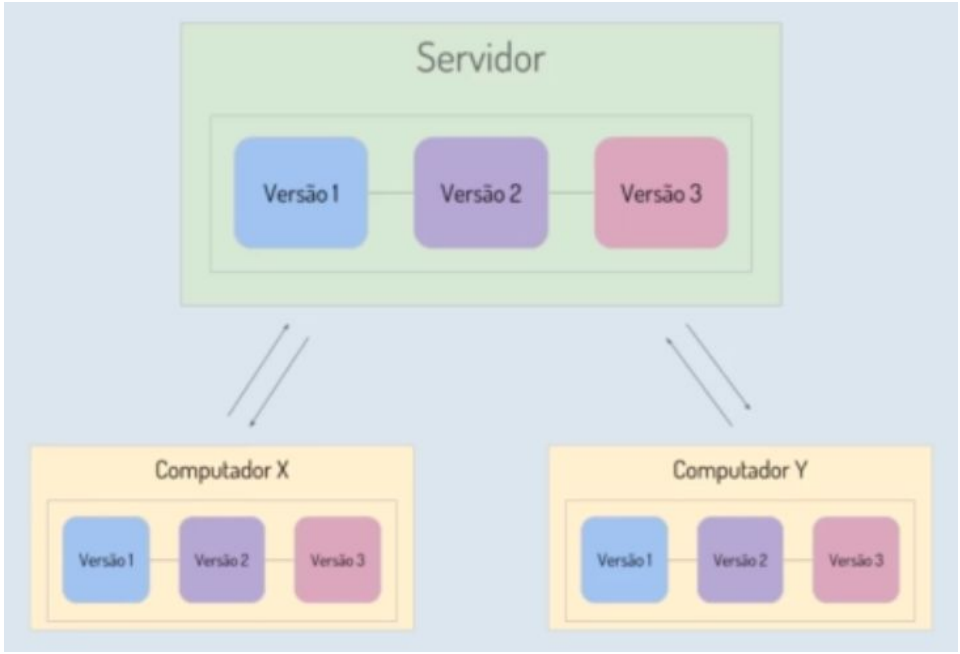
* '''Confiança''': O uso de repositórios remotos ajuda a não perder arquivos por eventos imponderáveis. Além disso e disponível fazer novos projetos sem danificar o desenvolvimento.

<ref>https://blog.wkm.com.br/o-que-%C3%A9-e-porque-usar-um-sistema-de-controle-de-vers%C3%A3o-23f00b08e12d</ref>

Versionamento de código com Git

- **Tipos de sistema de controle de versão:**





SUBVERSION

git

Versionamento de código com Git

- **GIT:**
  - **Criado por Linus Torvalds (2005): Código Aberto X BitKeeper**
  - **Git não é GitHub!**
  - **Utilidades:**
    - **Histórico:** O que mudou? Onde um bug surgiu? Temos um backup?
    - **Trabalho em Equipe:** Tarefas em paralelo, independentes ou colaborativas.
    - **Ramificação:** Possibilidade de ramificar e unir funcionalidades.
    - **Rastreabilidade:** Quem mexeu, porquê e em quê?
  - **Funcionalidades:**
    - **Operações Locais:** Git foi projetado para ser distribuído.
    - **Somente adição de alterações:** Remoções também contam.
    - **Integridade histórica:** História do projeto com detalhes.
    - **Autonomia:** Desenvolvedores trabalham com mínimas dependencias de código (evitar “deadlock” na equipe)

Versionamento de código com Git

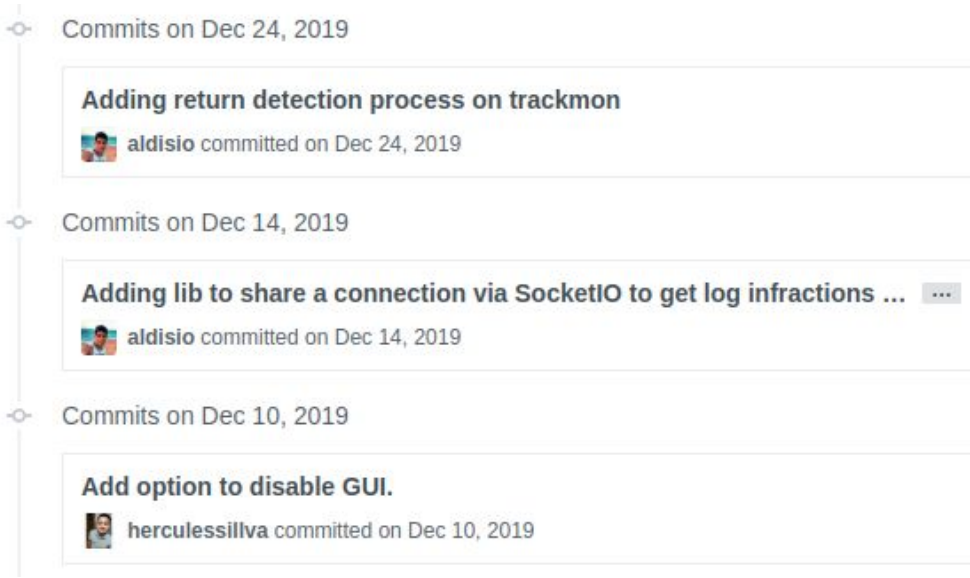
- **Instalação:**
  - Funciona tanto no Linux, quanto no Mac e até no Windows!
    - **Linux (ubuntu):**
      - `$ sudo apt-get install git`
      - Distros: https://git-scm.com/download/linux
    - **Outros SOs:**
      - https://git-scm.com/downloads

  - Checar se foi instalado com sucesso (Ver a versão):
    - `$ git --version`

Versionamento de código com Git

- **Passo 1: Diga para o git quem é você!**
  - `$ git config --global user.name “Fulano de tal”`
  - `$ git config --global user.email “fulanodetal@gmail.com”`
  - `$ git config --list`

```
aldisio@Avell-A62-MUV:~/github/trackmon$ git log -n 3
commit 3b11a06e3a98e86641d25ce333a51e56884c2508 (HEAD -> d
Author: Aldísio Medeiros <aldisiog@gmail.com>
Date:   Tue Dec 24 09:36:39 2019 -0300

    Adding return detection process on trackmon

commit ee5e1cf026fe10f1d1913f0893ad6f63e7a448eb
Author: Aldísio Medeiros <aldisiog@gmail.com>
Date:   Sat Dec 14 18:16:30 2019 -0300

    Adding lib to share a connection via SocketIO to get l

commit a6d550f97286306a3e4391767efa3c64ac295acc
Author: herculessillva <herculessilva@lapisco.ifce.edu.br>
Date:   Tue Dec 10 16:12:50 2019 -0300

    Add option to disable GUI.
```



Versionamento de código com Git

- **Passo 2: Inicialize seu repositório (GIT INIT)**
  - `$ git init`
  - A pasta do repositório **poderá ou não estar vazia.**
  - Se estiver preenchida, poderá versionar os arquivos contidos a partir daquela data.
  - O repositório é criado com uma ramificação (branch) padrão chamada de ***master***
  - Observe que **será criado um diretório oculto** chamado de ***.git.***

```
aldisio@Avell-A62-MUV:~/github$ mkdir trainner-mod1
aldisio@Avell-A62-MUV:~/github$ cd trainner-mod1
aldisio@Avell-A62-MUV:~/github/trainner-mod1$ ls -la
total 8
drwxr-xr-x  2 aldisio aldisio 4096 mar  3 02:21 .
drwxr-xr-x 60 aldisio aldisio 4096 mar  3 02:21 ..
aldisio@Avell-A62-MUV:~/github/trainner-mod1$ git init
Initialized empty Git repository in /home/aldisio/github/trainner-mod1/.git/
aldisio@Avell-A62-MUV:~/github/trainner-mod1$ ls -la
total 12
drwxr-xr-x  3 aldisio aldisio 4096 mar  3 02:21 .
drwxr-xr-x 60 aldisio aldisio 4096 mar  3 02:21 ..
drwxr-xr-x  7 aldisio aldisio 4096 mar  3 02:21 .git
aldisio@Avell-A62-MUV:~/github/trainner-mod1$
```

Versionamento de código com Git

- **Passo 3: Adicionando arquivos a serem rastreados (GIT ADD)**
  - `$ git add README.md (para todos o arquivo README.md)`
  - `$ git add [-A,--all, .] (para todos os arquivos)`

```
aldisio@Avell-A62-MUV:~/github/trainner-mod1$ ls -l
total 0
aldisio@Avell-A62-MUV:~/github/trainner-mod1$ date --date=now>README.txt
aldisio@Avell-A62-MUV:~/github/trainner-mod1$ cat README.txt
ter mar  3 02:49:13 -03 2020
aldisio@Avell-A62-MUV:~/github/trainner-mod1$ git status
On branch master

No commits yet

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

        README.txt

nothing added to commit but untracked files present (use "git add" to track)
aldisio@Avell-A62-MUV:~/github/trainner-mod1$ git add README.txt
aldisio@Avell-A62-MUV:~/github/trainner-mod1$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   README.txt

aldisio@Avell-A62-MUV:~/github/trainner-mod1$
```

Versionamento de código com Git

- **Passo 3: Acompanhando atualizações nos arquivos rastreados (GIT STATUS):**

```
aldisio@Avell-A62-MUV:~/github/trainner-mod1$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   README.txt

aldisio@Avell-A62-MUV:~/github/trainner-mod1$ date --date=now > README.txt
aldisio@Avell-A62-MUV:~/github/trainner-mod1$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   README.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:   README.txt

aldisio@Avell-A62-MUV:~/github/trainner-mod1$
```

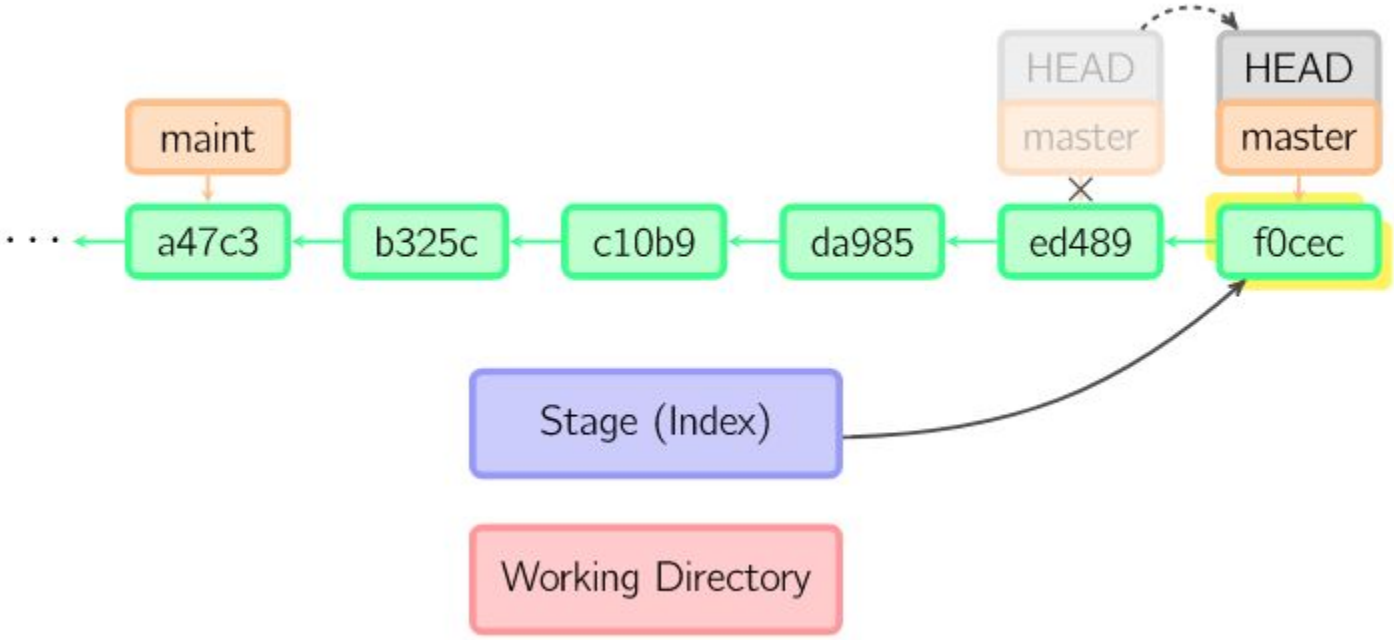
Versionamento de código com Git

- **Passo 4: Enviando alterações para o repositório local (GIT COMMIT):**
  - `$ git commit -m “Adding README.txt file”`

```
aldisio@Avell-A62-MUV:~/github/trainner-mod1$ git add README.txt
aldisio@Avell-A62-MUV:~/github/trainner-mod1$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   README.txt

aldisio@Avell-A62-MUV:~/github/trainner-mod1$ git commit -m 'Adding README.txt file'
[master (root-commit) 2b7d7c2] Adding README.txt file
 1 file changed, 1 insertion(+)
 create mode 100644 README.txt
aldisio@Avell-A62-MUV:~/github/trainner-mod1$ git status
On branch master
nothing to commit, working tree clean
aldisio@Avell-A62-MUV:~/github/trainner-mod1$
```

Versionamento de código com Git

- **Passo 4: Enviando alterações para o repositório local (GIT COMMIT):**
  - `$ git commit -m “Adding README.txt file”`



Versionamento de código com Git

- **Lembre-se: “Commits menores geram conflitos menores”**

Versionamento de código com Git

- **Logs: Listando o histórico de commits (GIT LOG):**
  - `$ git log`
  - `$ git log --oneline`

```
aldisio@Avell-A62-MUV:~/github-trainner/cobaia02-repository$ git log
commit 930b707ec7151a744bba782102e58b1bf4726fc3 (HEAD -> master)
Author: Aldísio Medeiros <aldisiog@gmail.com>
Date:   Sun Mar 8 10:31:29 2020 -0300

    Third commit

commit 6f63bc46d746fec11d5dbd260e8095c77636ac2b
Author: Aldísio Medeiros <aldisiog@gmail.com>
Date:   Sun Mar 8 10:31:03 2020 -0300

    Second commit

commit 4e816c8287b52c9aec4ad0a13aa52a5126fb87fa (origin/master)
Author: Aldísio Medeiros <aldisiog@gmail.com>
Date:   Sat Mar 7 19:07:30 2020 -0300

    Initial commit
```

```
aldisio@Avell-A62-MUV:~/github-trainner/cobaia02-repository$ git log --oneline
930b707 (HEAD -> master) Third commit
6f63bc4 Second commit
4e816c8 (origin/master) Initial commit
```

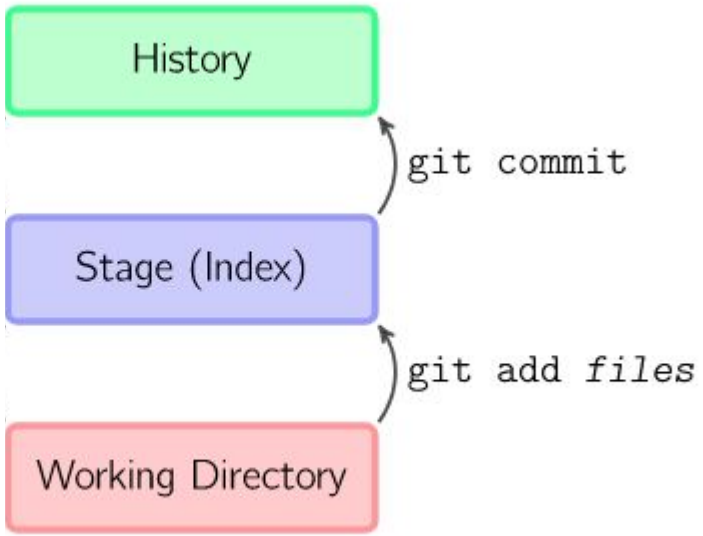
Versionamento de código com Git

- **Logs: Listando o histórico de commits (GIT LOG):**
  - `$ git log --author=aldisiog@gmail.com`
  - `$ git log --grep=Second`
    - `Mas e se o “Second” estiver todo em minúsculo?`

```
aldisio@Avell-A62-MUV:~/github-trainner/cobaia02-repository$ git log --author=aldisiog@gmail.com --grep=Second
commit 6f63bc46d746fec11d5dbd260e8095c77636ac2b
Author: Aldísio Medeiros <aldisiog@gmail.com>
Date:   Sun Mar 8 10:31:03 2020 -0300

    Second commit
```

Versionamento de código com Git

- **Mas como funciona o GIT?**
  - **Working directory:** Diretório onde as mudanças são primeiramente realizadas, é a pasta onde são alterados os arquivos locais.
  - **Stage (Index / Área de transição):** Área de preparação das mudanças antes de serem gravadas (commitadas no repositório). Contém um snapshot dos **arquivos rastreados.** Mantém um **snapshot** do repositório no estágio do último commit.
  - **History (commited):** Mudanças que foram gravadas no diretório de versionamento. Existe um ponteiro que indica a qual commit estamos chamado de **HEAD** (veremos adiante)



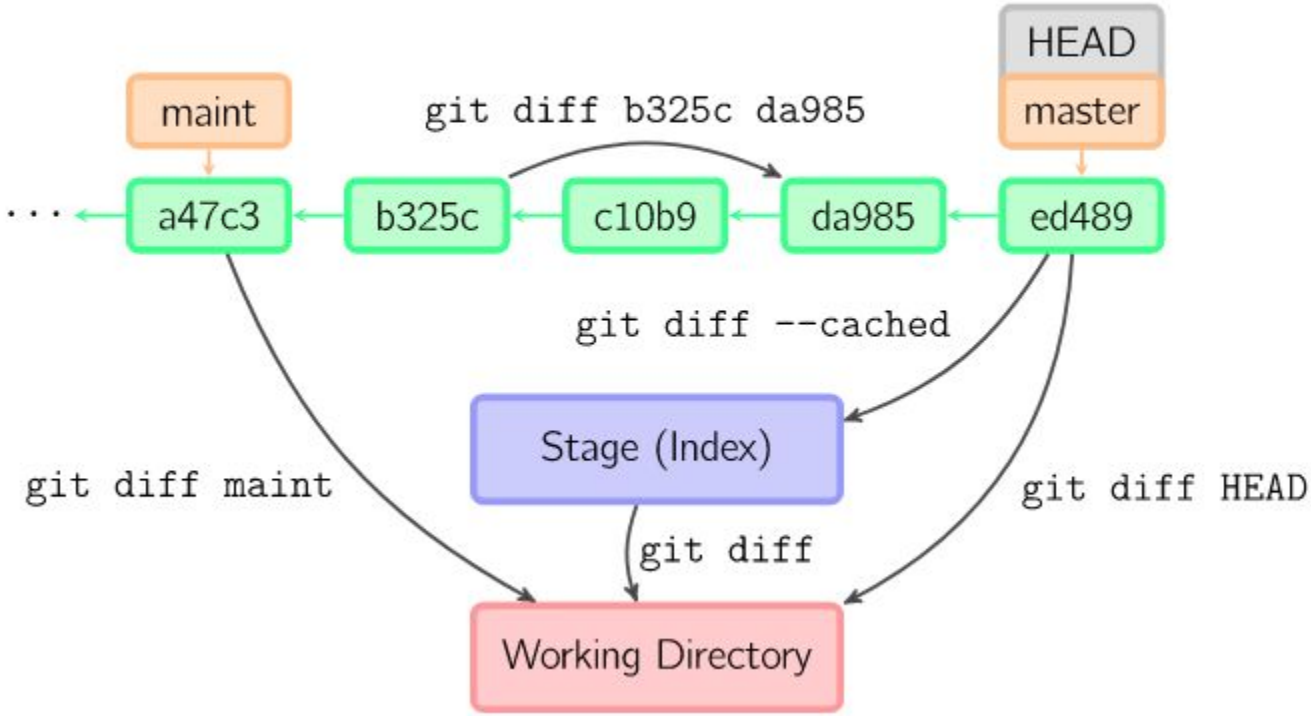
Versionamento de código com Git

- **Comando para checagem/comparação de mudanças (GIT DIFF):**
  - **Na prática, explicar diferenças entre:**
    - `$ git diff`
    - `$ git diff --cached`
    - `$ git diff HEAD`
      - **O que é o HEAD??**
    - **Passo 1: Criar arquivo file1.txt e adicionar no stage.**
    - **Passo 2: Alterar arquivo file2.txt e ver diferenças.**
    - **Passo 4: Commitar apenas o que já estava no stage.**
    - **Passo 5: Ver diferenças dos git diff.**

  - **Quais mudanças nos comandos:**
    - **Git diff, git diff --cached e git diff HEAD?**

Versionamento de código com Git

- **Comando para checagem/comparação de mudanças (GIT DIFF):**



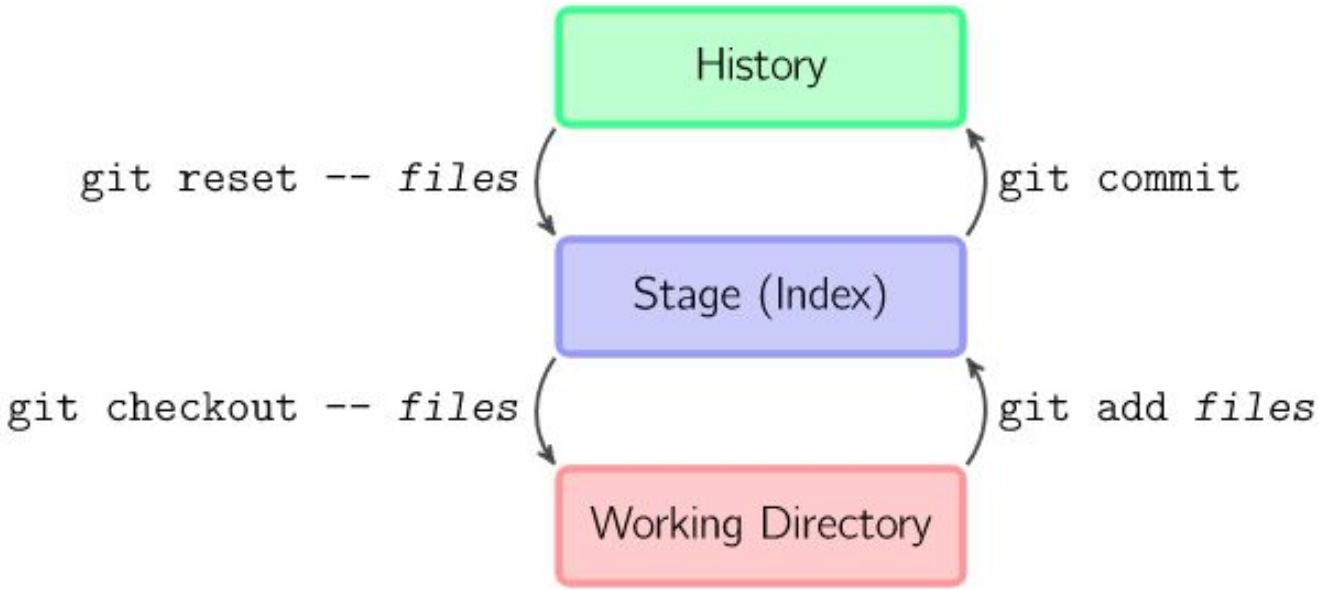
Versionamento de código com Git

- **Voltando no tempo… (GIT CHECKOUT):**
  - Um bug foi identificado na sua versão local, o que fazer para restaurar?
    - **Opção 1:** Analisar todas as mudanças no repositório.
    - **Opção 2:** Retornar a uma versão anterior a identificação do bug e analisar apenas os arquivos que mudaram na versão seguinte ao surgimento do bug.

  - **Desfazendo alterações indesejadas / exclusão acidental de arquivos.**
    - `$ git checkout arquivo`: restaura o arquivo ao estado em que estava no último commit.
    - `$ git checkout .` : restaura todos os arquivos para o estado no último commit.
    - `$ git checkout HASH-COMMIT`: restaura para um commit específico.

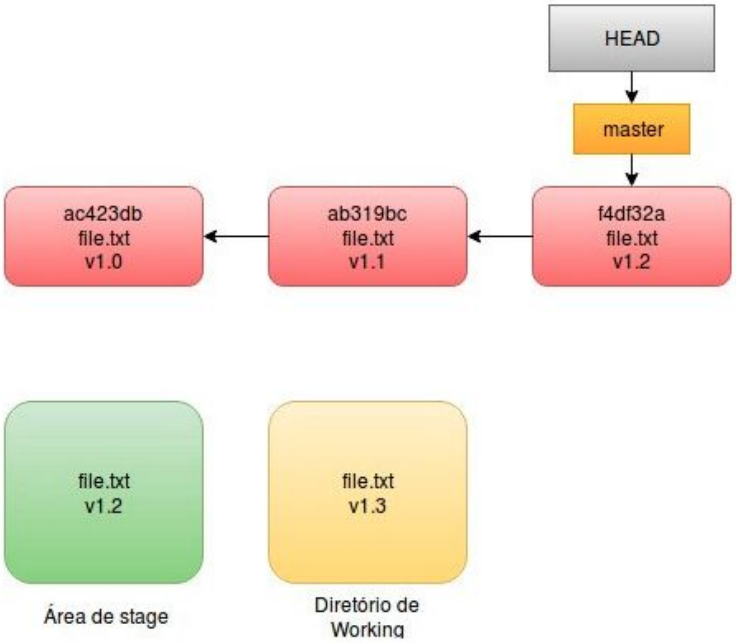
Versionamento de código com Git

- **Voltando no tempo… (GIT CHECKOUT / RESET):**
  - Vamos ver na prática como isso acontece.



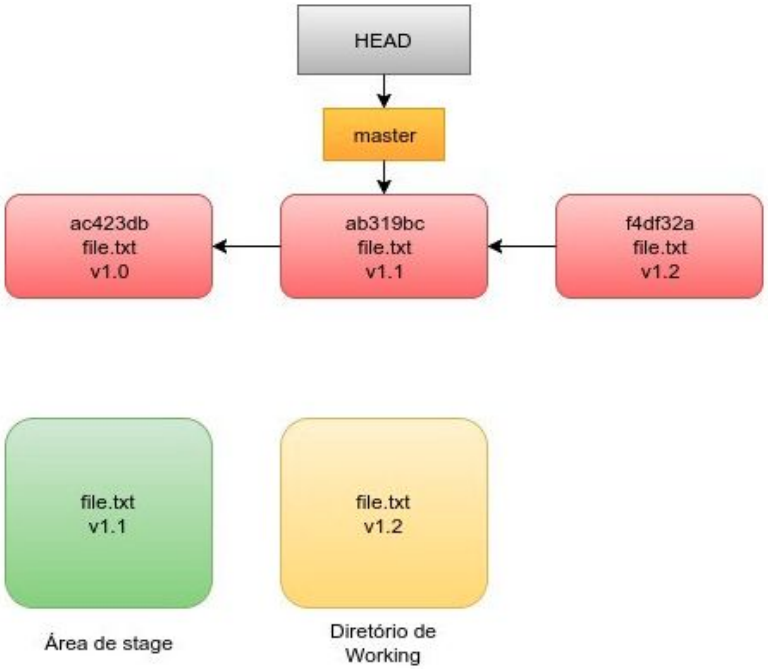
Versionamento de código com Git

- **Desfazendo alterações nos outros estágios do git**
  - **No diretório de trabalho (checkout), na stage (reset HEAD).**



Situação após o commit f4df32a:

```
$ git commit -m “v1.2”
```

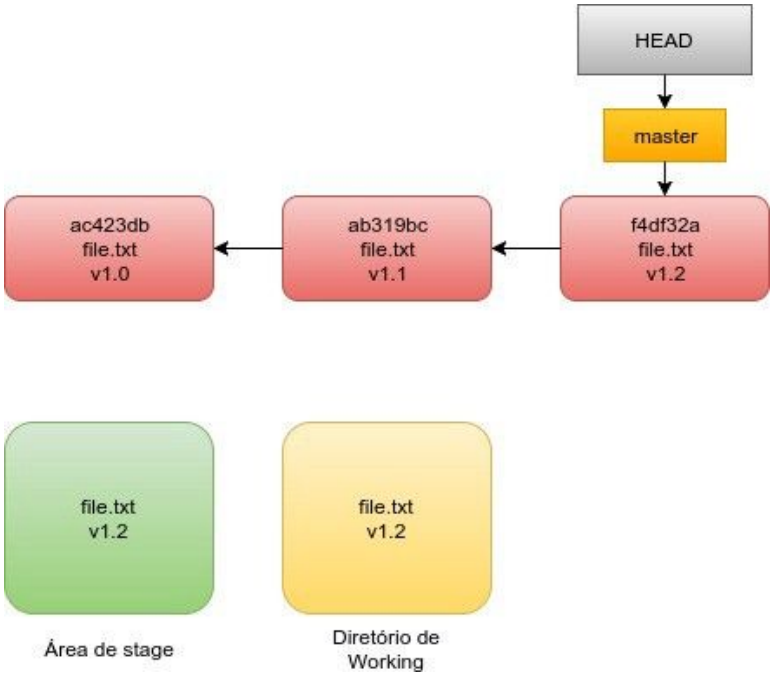


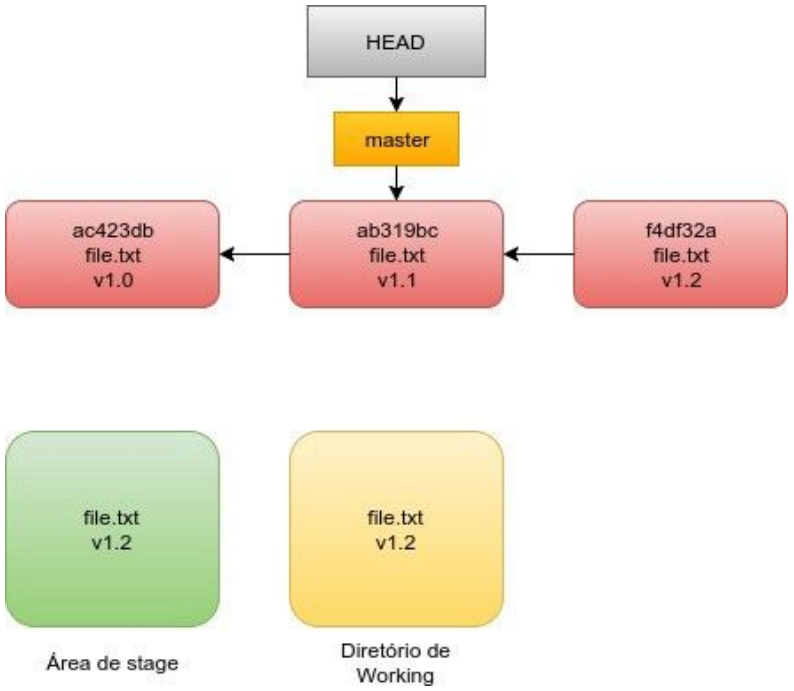
Situação após o reset:

```
$ git reset HEAD^
```

Versionamento de código com Git

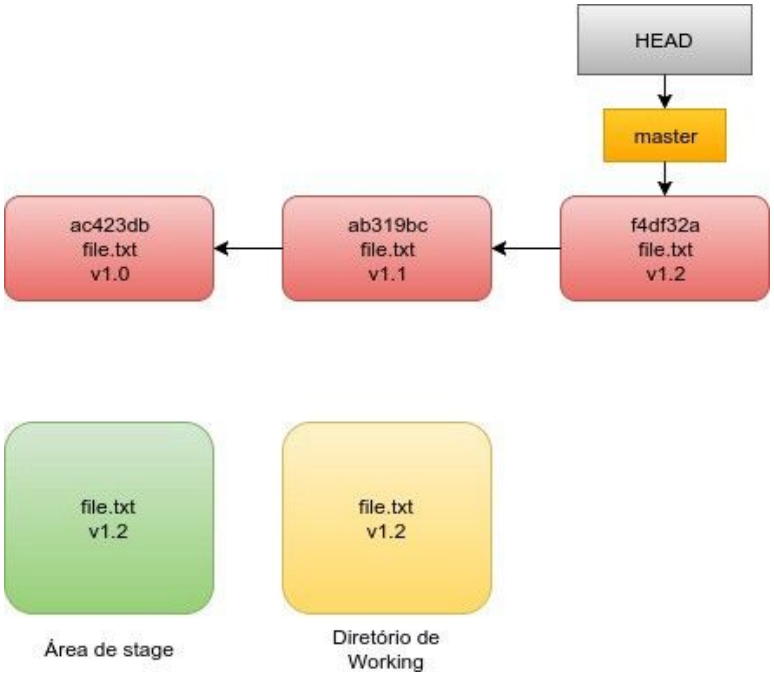
- **Cuidados com o git reset:**
  - **-- soft:** `$git reset --soft HEAD^`
  - **-- mixed:**
  - **-- hard:**
    - (^ e ~ tem efeitos parecidos mas não iguais)

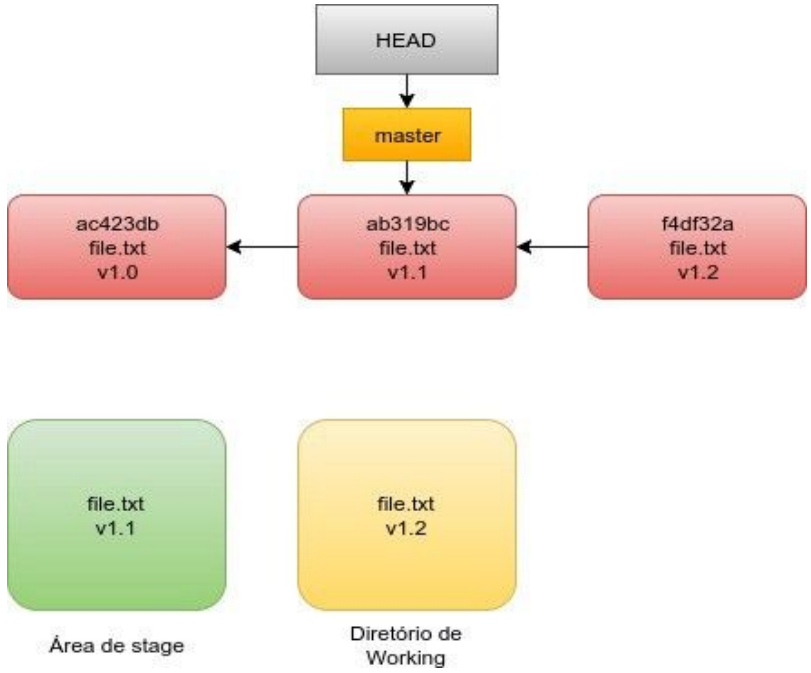




Versionamento de código com Git

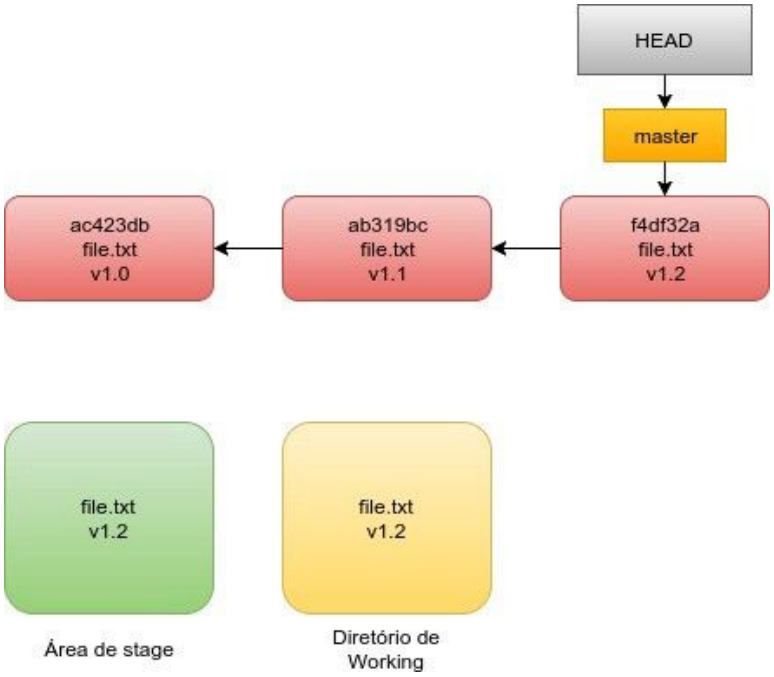
- **Cuidados com o git reset:**
  - **-- soft: `$git reset --soft HEAD^`**
  - **-- mixed: `$git reset --mixed HEAD^` ou `git reset HEAD^`**
  - **-- hard:**
    - (^ e ~ tem efeitos parecidos mas não iguais)

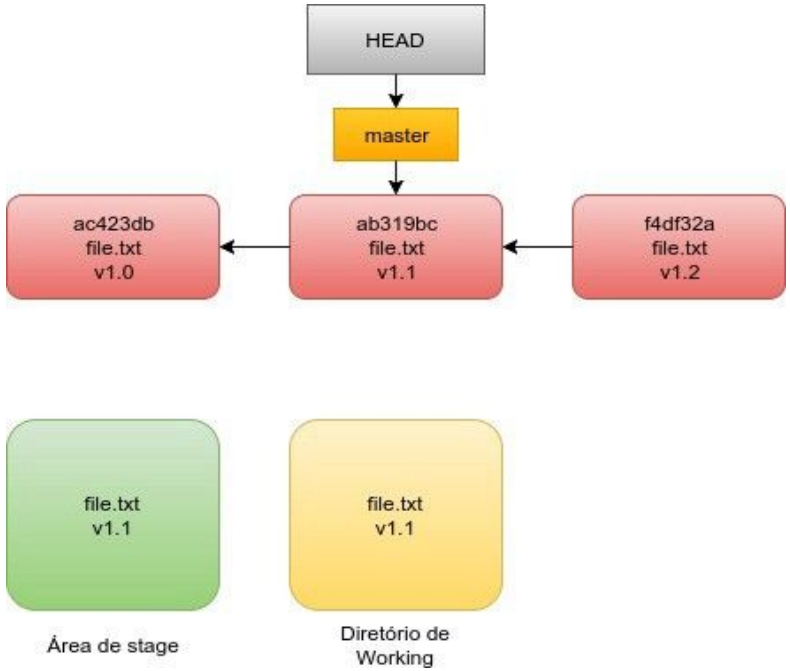




Versionamento de código com Git

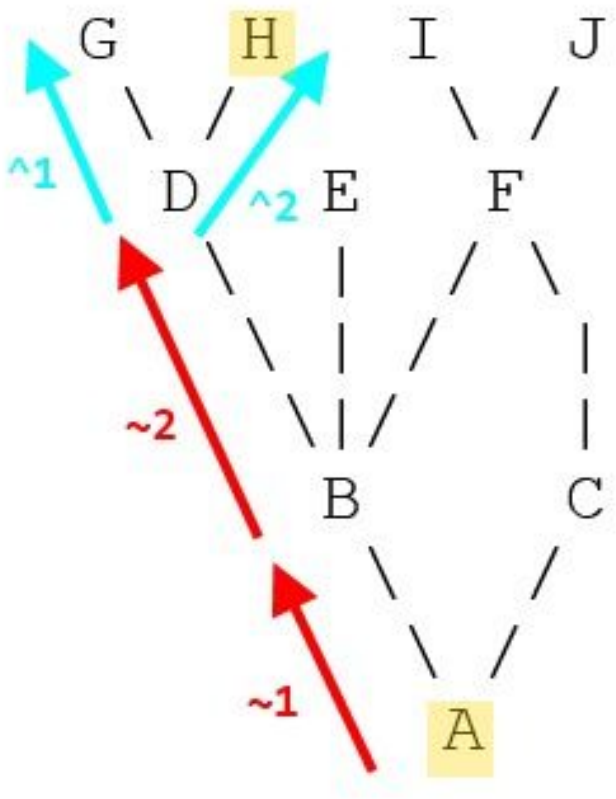
- **Cuidados com o git reset:**
  - **-- soft: `$git reset --soft HEAD^`**
  - **-- mixed: `$git reset --mixed HEAD^` ou `git reset HEAD^`**
  - **-- hard: `$git reset --hard HEAD^` (Use com atenção)**
    - (^ e ~ tem efeitos parecidos mas não iguais)





Versionamento de código com Git

- **Cuidados com o git reset:**
  - ^ e ~ tem efeitos parecidos mas não iguais



H = A~2^2

Versionamento de código com Git

- **Vamos ver as diferenças entre o ponteiro HEAD e o diretório de trabalho!**
  - Podemos manipular o ponteiro HEAD **para fins de comparações** com o arquivo no diretório atual.
  - Esta funcionalidade pode ser **útil para comparar mudanças em um mesmo código** com relação aos últimos commits realizados!

- **Pergunta**: Para ver as diferenças, utilizamos qual comando git ?
- **Pergunta**: Para voltarmos o HEAD para commits anteriores, utilizando qual comando git ?

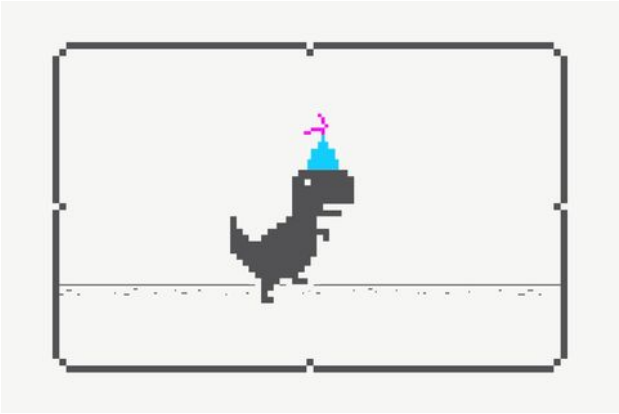
Versionamento de código com Git

- **Deixar de rastrear um arquivo (GIT RM --CACHED FILE]**
  - `$ git rm --cached file.txt`

- Cenário: Imagine que você criou um arquivo chamado **config.py** para manter as variáveis de configuração do seu software local. Após criar e configurar, você adicionou ele no stage via comando **"git add config.py"**. Contudo, antes de commitar você conversou com a equipe e descobriu que um colega estava organizando todas as configurações locais em um banco de dados, não mais em arquivo e este será o padrão para o projeto a partir de então. Agora, você precisa descartar o arquivo config.py, mas ele já está no stage, o que fazer?

Versionamento de código com Git

- **E se cair a internet, como trabalha (GIT CLONE)?**



- **Comando git clone**
  - O clone de um repositório traz consigo não só o código em sua **última versão de commit** mas toda a história do repositório (default master)
    - `$ git clone git@github.com:aldisio/cobaia03-repository.git`
  - Mas e se eu quiser clonar de uma branch específica?
    - `$ git clone -b new-feature --single-branch git git@github.com:lapisco/facerec.git`

Versionamento de código com Git

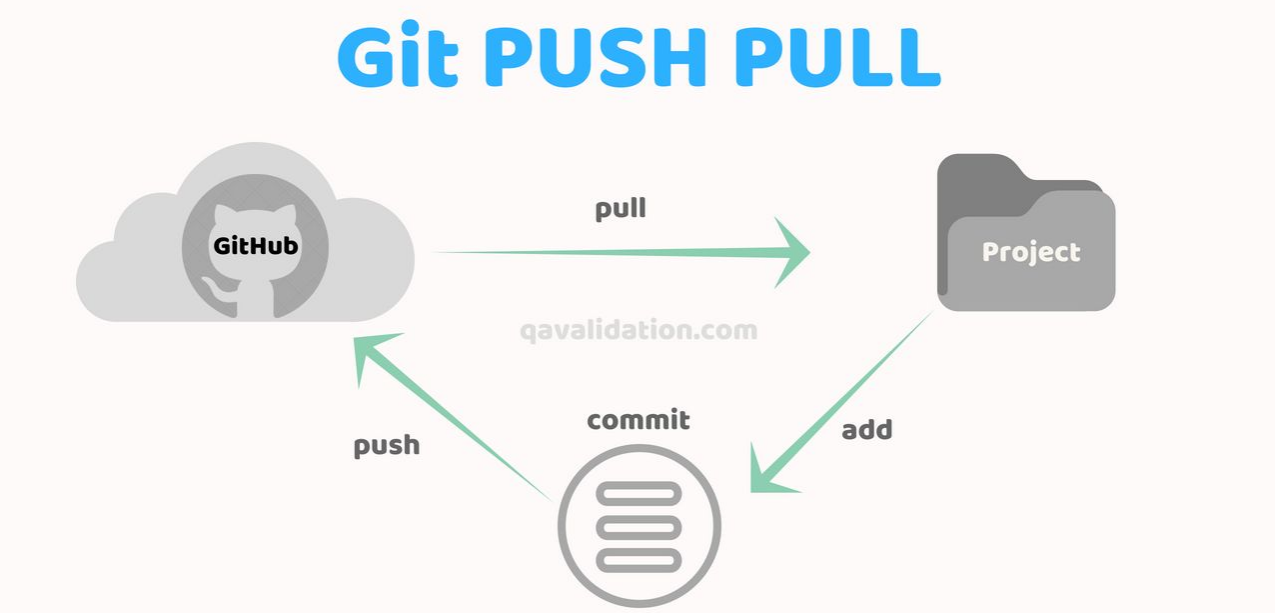
- **Comando git pull (GIT PULL)**

  - Após as alterações que acontecem no repositório remoto, precisamos atualizar nosso diretório de trabalho com estas mudanças, ocasionadas por terceiros (diretório público ou privado para equipe) ou por você mesmo, porém feitas de outro repositório (de um outro PC, por exemplo)
    - `$ git pull`

  - O git pull incorpora alterações de um repositório remoto **na branch atual,** combina dois comando em sequência.
    - `$ git fetch seguido de um`
    - `$ git merge FETCH_HEAD ( git -rebase`

Versionamento de código com Git

- **Comando git push (GIT PUSH)**
  - Envia as alterações gravadas no repositório local para o repositório remoto (se houver remote configurado e ativo ele pode ser omitido)
    - `$ git push [remote] [branch]`
  - **É importante no dia a dia de trabalho, executar o `git pull`** (no início dos trabalhos) e com o `git push` (ao final do trabalho).
    - **Vejamos isso na prática**!

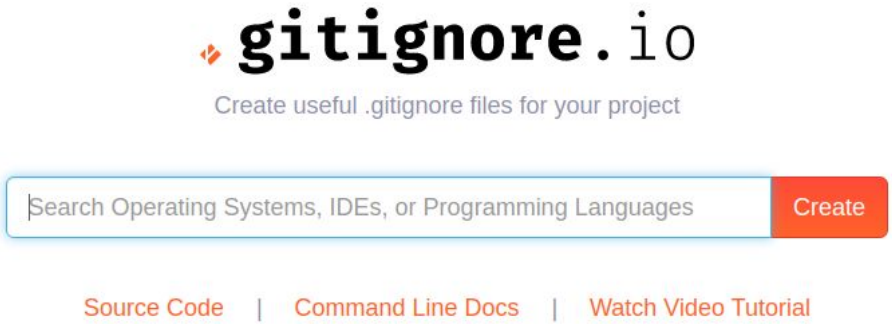


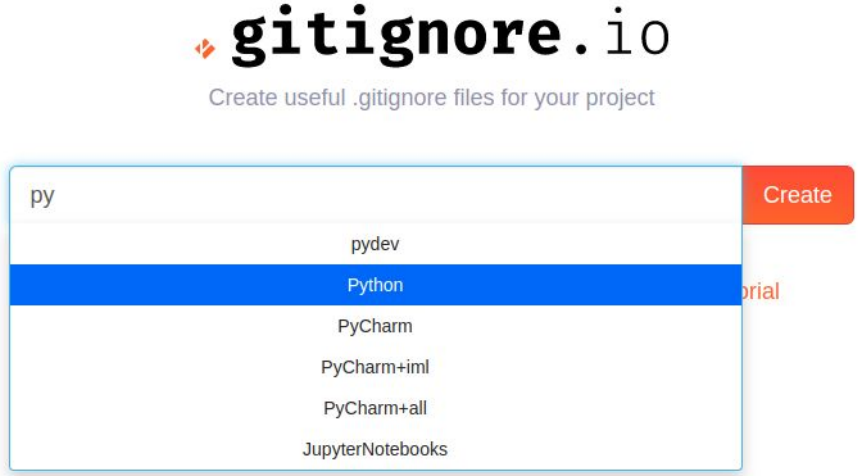
Versionamento de código com Git

- **Ignorando arquivos com .gitignore.**
  - Arquivos de cache local, com senhas, de testes (vídeos, fotos, csv, txt de log, etc)
  - Arquivos de configuração de IDE
- **Entendendo a sintaxe do .gitignore:**
  - **Utilize globs!** (*.bmp ; directory_root/*.log)
    - **E se eu quiser ignorar todos os *.jar mas quiser que um arquivo jar específico suba para o repositório?**
    - **Vejamos isso na prática**!
- Lembre-se que **a ordem das declarações é importante!**
- **Mas, quais tipos de arquivos devem ignorados pelo repositório?**
  - Arquivos compilados, como **.so** e **.class**;
  - Dependências gerenciadas por **gerenciadores de dependências**, como o package-lock.json, em uma aplicação em React JS, por exemplo);
  - Arquivos **temporários de build** (executáveis em um projeto em C/C++)
  - **Arquivos privados** locais (como por exemplo, arquivos que guardam dados secretos no container de injeção de dependências);
  - **Arquivos** pessoais **salvos pela IDE**;

Versionamento de código com Git

- **Exemplos prontos de .gitignore:**
  - https://github.com/github/gitignore
- **Ferramenta de criação rápida de .gitignore:**
  - **Gitignore.io:** https://gitignore.io/





Versionamento de código com Git

- **Comando git remote**
  - Mas e se eu quiser fazer alterações e armazená-las no repositório remoto?
    - Solução: Adicione um repositório remoto!
  - Comando:
    - `$ git remote add [nome do remote] [URL do remote]`
    - `$ git remote add **origin** git@github.com:aldisiomedeiros-lab/cobaia02-repository.git`
  - Depois que o repositório remote estiver adicionado, quando for fazer o pull para o repositório, execute a linha informando o repositório remoto e a branch.
    - `$ git push **origin master**`

Conhecendo o GitHub

Versionamento de código com Git

- **GITHUB (Similares: BitBucket e GitLab)**
  - **Plataforma web** para armazenamento e compartilhamento de repositórios públicos ou privados.
  - **Criada em 2008** (3 anos após o nascimento do Git) e vendida por 7.5 Bi para microsoft no início de 2018.
    - Possibilita:
      - Cadastro de **ISSUES** (questões, bugs)
      - Histórico e **comparação** de commits.
      - **Estatísticas** de versões do código.
      - Gestão do **acesso ao código**.
      - Gerar **tags** e **releases** do código.
  - No plano gratuito você pode criar **repositórios públicos e/ou repositórios privados** (limitados a 3 colaboradores)
  - **Repositório públicos são para consulta**, alterações precisarão ser aceitas (Erro 403)
    - Vamos ver isso na prática!

Versionamento de código com Git

- **Star, watching**
  - Ver no github.

- **Fork: Uma forma de bifurcar um repositório.**
  - O fork funciona de forma semelhante ao clone, porém **o fork preserva o vínculo de "herança"** entre a sua cópia e o repositório original.
    - Isto pode ser observado no histórico de commits!
  - Além de copiar todo o repositório, o fork **possibilita modificações no repositório de forma direta** diferentemente do clone.

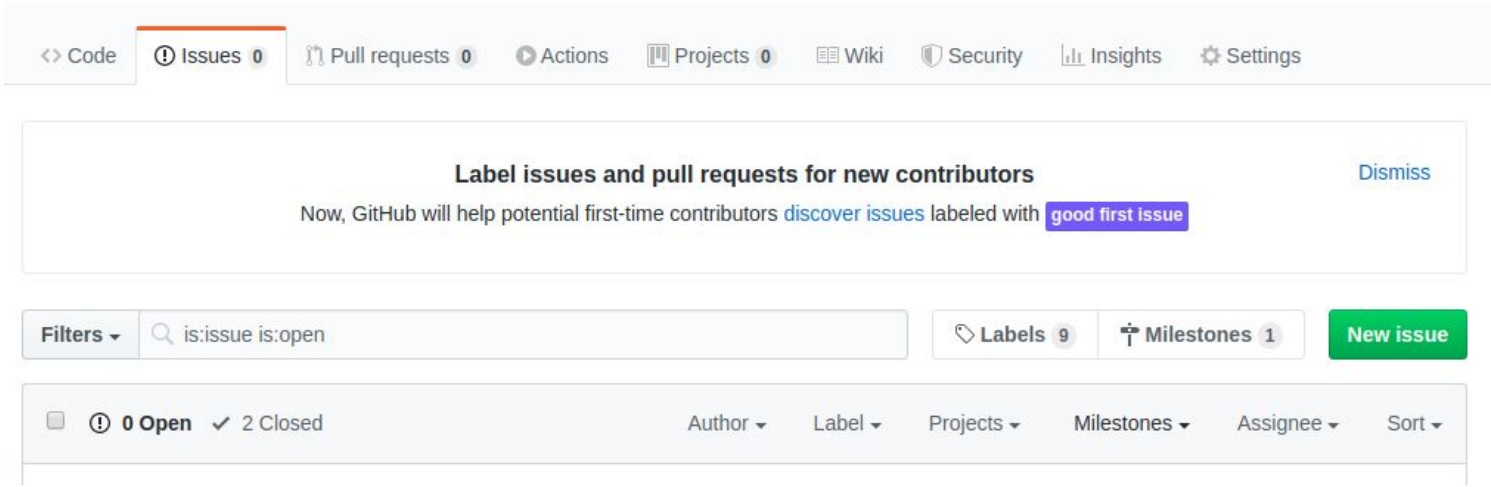
- **Pull request: Uma forma de colaborar com os repositórios de terceiros.**
  - **Repositórios públicos não é sinônimo de que qualquer um pode mexer,** a qualquer momento e de qualquer forma. (Isso produziria muitos bugs!)
  - A funcionalidade pull request é uma forma de **informar ao administrador** que você está **sugerindo juntar suas mudanças** com o repositório original.
  - Pull request funciona **também entre *branchs*.**
  - Esta funcionalidade **possibilita interação** entre os diferentes devs até o aceite.

Versionamento de código com Git

- **Resumindo o processo de contribuição via fork:**
  1. **Fork do projeto para o seu usuário.**
  2. **Clone do projeto 'forkado' na sua máquina.**
  3. **Taca-le pau ..**
  4. **Atualização do seu projeto com o projeto original.**
  5. **Commitar e enviar suas alterações para o seu GitHub.**
  6. **Abrir um pull request para o projeto original.**
- **MAS….**
  1. **E se no meio do percurso entre terminar de criar suas mágicas na versão local um outro dev adicionou novos paranauês no repositório original via Pull Request??**
  2. **Adicione o remote UPSTREAM!**
     - `$ git remote add upstream [URL repo original]`
     - `$ git fetch upstream`
     - `$ git checkout master (garantir que está na master)`
     - `$ git merge upstream/master master`
     - `$ git push -f origin master (git commit -am '...', antes)`

Versionamento de código com Git

- **ISSUES**
  - Possibilidade de terceiros notificar bugs, questionar, sugerir melhorias
- **LABELS:**
  - Possibilita categorizar as issues cadastradas pelos colaboradores/usuários do repositório.
- **MILESTONES**
  - Releases futuras daquele repositório.



- Nós podemos **contribuir com a resolução das issues** por meio de pull requests de resolução/correção!

Versionamento de código com Git

- **README.md**
  - Arquivo didático com instruções importantes sobre seu repositório.
  - Permite formatação utilizando markdown (https://dillinger.io/)

Recorder streaming to file:

Run the python saveStreamingYOLO.py script to recorder video streaming to a video file, identifing the object of interest by the YOLO model (This model was trainned by VOC dataset)

Requirements

- Python3
- Numpy
- OpenCV
- Darkflow
- tensorflow-gpu: 1.12.0

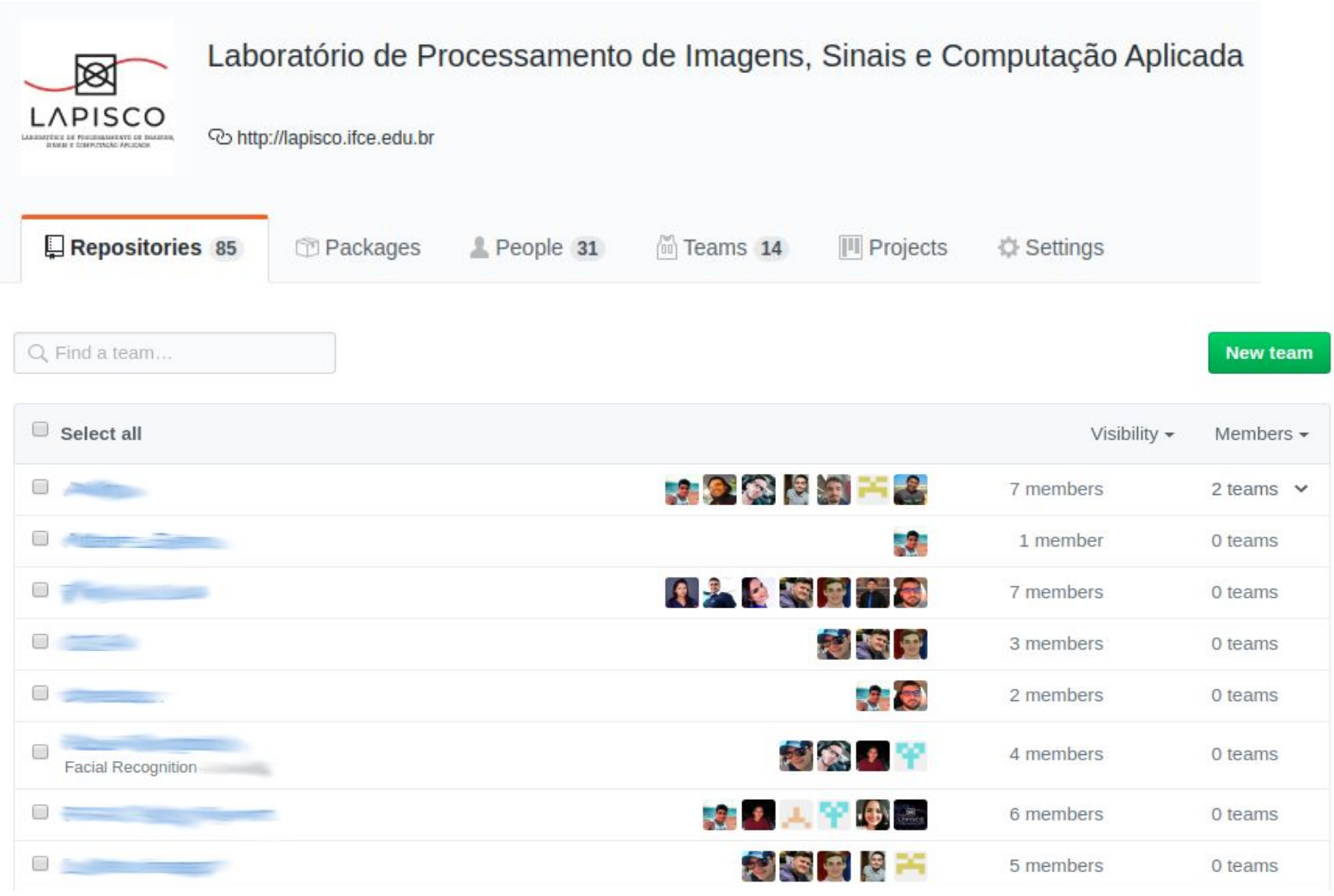
You need download the model files with weights, Yolo_VOC.zip: Download. After, unzip this directory into root directory, i.e, same level where is saveStreamingYOLO.py.

Usage:

```
$ python3 saveStreamingYOLO.py 10.102.1.151 user password 0 30 /mnt/Dados/videos_cameras/portaria_ifce_fro
```

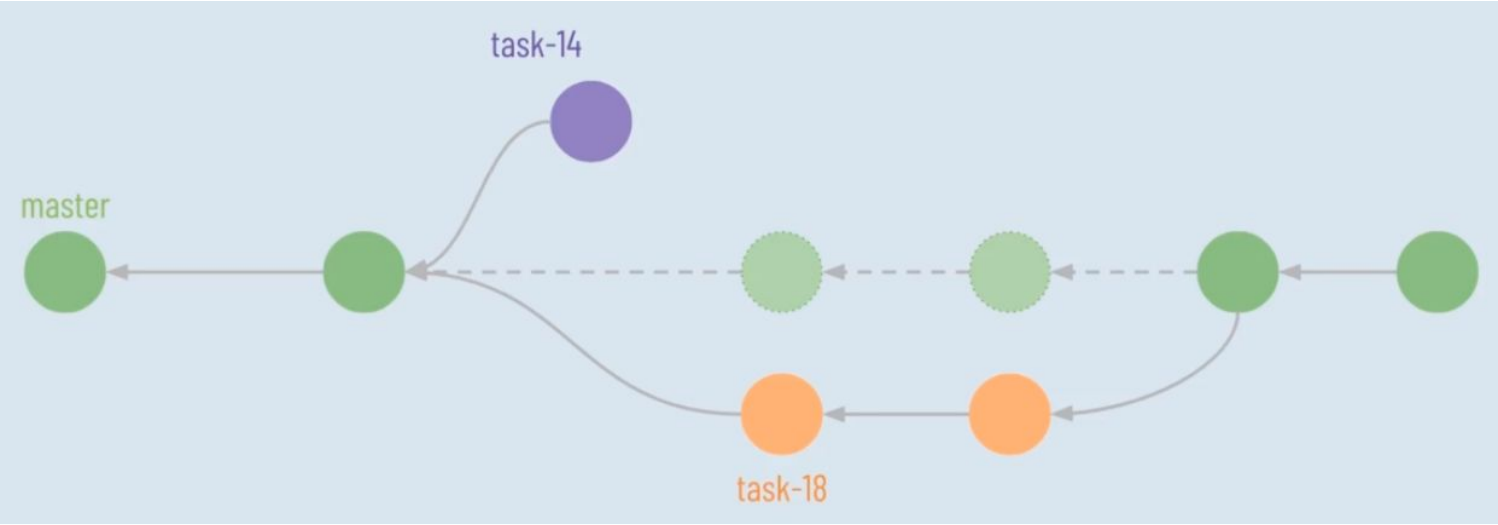
Versionamento de código com Git

- **Criação de equipes e repositórios específicos.**



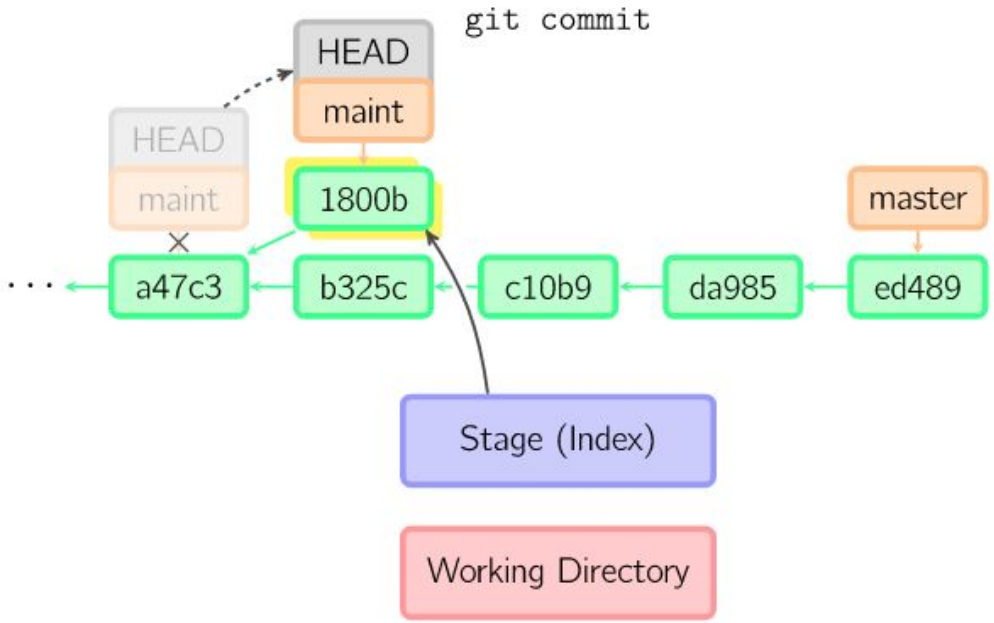
Versionamento de código com Git

- **Branches**
  - É literalmente uma ramificação da base do código.
    - São bifurcações dentro de um mesmo repositório (não confundir com o fork!)
  - As **branches** possibilitam a colaboração de equipes de desenvolvedores atuarem em diferentes features de um mesmo projeto.
  - Enquanto mantemos o código de produção na **branch principal** (por exemplo), podemos criar uma **ramificação para trabalhar livremente** neste código e **somente depois juntar** (“mergear”) estes códigos.



Versionamento de código com Git

- **Branches**
  - Cada branch cria uma linha de commits própria que poderá ser posteriormente agregada ao histórico central (master) de alterações.



Versionamento de código com Git

- **Branches**
  - Listando as branches locais:
    - `$ git branch`
  - Criando uma nova branch local:
    - `$ git branch develop`
  - Alternando para uma branch:
    - `$ git checkout develop`
  - Atalho para os dois comandos anteriores:
    - `$ git checkout -b develop`
  - Enviando para o repositório remoto
    - `$ git push origin develop`
  - Removendo branch local:
    - `$ git branch -d develop2`
  - Removendo branch do repositório remoto
    - `$ git push --delete origin develop2`

Versionamento de código com Git

- **Juntando mudanças em uma mesma branch (GIT MERGE):**
  - `$ git checkout develop`
  - `…. alterações ….`
  - `$ git add .`
  - `$ git commit -m ‘mudancas na develop’`
  - `$ git checkout master`
  - `$ git merge develop`

- **Pergunta: Depois das alterações, antes do “git add” já poderíamos ter feito um git checkout para a master?**

Versionamento de código com Git

- **Assim como na vida, no git algumas vezes ocorrem conflitos:**

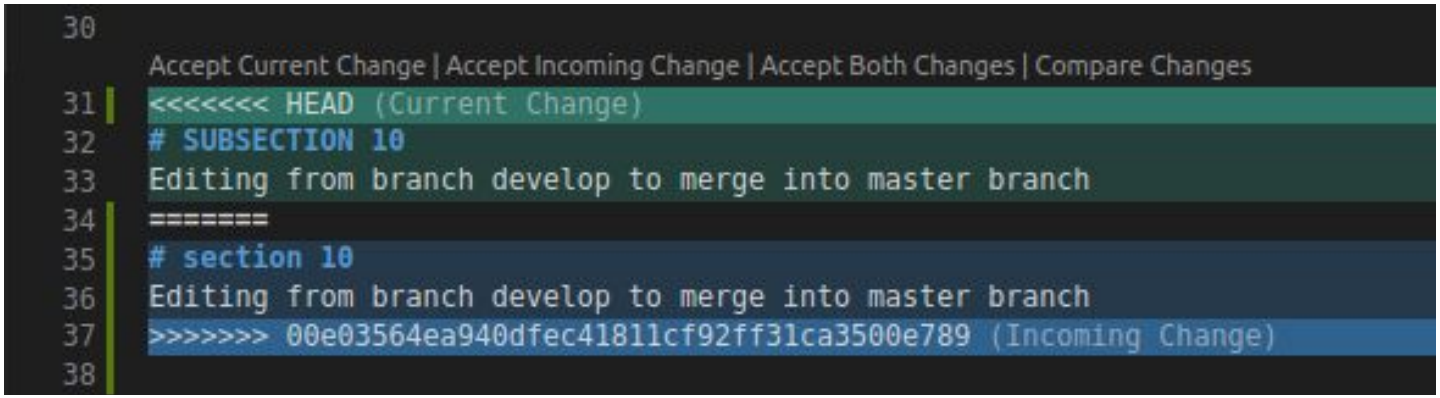
```
<!-- Author: Alexander Shvets (alex@githowto.com) -->
<html>
  <head>
<<<<<<< HEAD
    <link type="text/css" rel="stylesheet" media="all" href="style.css" />
=======
    <!-- no style -->
>>>>>>> master
  </head>
  <body>
    <h1>Hello,World! Life is great!</h1>
  </body>
</html>
```

- **Quando há conflitos, infelizmente o git não sabe qual alteração deve permanecer, neste caso se faz necessária análise manual.**

```
aldisio@Avell-A62-MUV:~/github-trainner/cobaia01-repository-fork$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From github.com:aldisio/cobaia01-repository
   d67fc6e..00e0356  master     -> origin/master
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

Versionamento de código com Git

- **No vscode (Outras IDEs também possuem esta funcionalidade)**



- **Após a resolução manual do conflito:**
  - **Adicione as alterações novamente no stage:**
    - `$ git add .`
  - **Commit as mudanças indicando que neste caso foi realizada uma correção de conflito**
    - `$ git commit -m 'Fixing conflits'`
  - **Suba as mudanças**
    - `$ git push origin master`

Versionamento de código com Git

- **Git stash**
  - Caso estejamos trabalhando em alguma feature e naquele exato momento precisaremos fazer checkout para outra branch mas não queremos commitar o que foi feito?
  - O git disponibiliza uma pilha na memória para guardar alterações temporárias e uma forma de acessá-la é por meio do git stash!
  - **Comando para criar uma stash:**
    - `$ git stash`
    - `$ git stash save “Saving temporary changes”`
  - **Comando para listar stash:**
    - `$ git stash list`

```
aldisio@Avell-A62-MUV:~/github-trainner/cobaia01-repository-fork$ git stash save 'temporary changes 01'
Saved working directory and index state On develop: temporary changes 01
aldisio@Avell-A62-MUV:~/github-trainner/cobaia01-repository-fork$ git stash save 'temporary changes 02'
Saved working directory and index state On develop: temporary changes 02
aldisio@Avell-A62-MUV:~/github-trainner/cobaia01-repository-fork$ git stash save 'temporary changes 03'
Saved working directory and index state On develop: temporary changes 03
aldisio@Avell-A62-MUV:~/github-trainner/cobaia01-repository-fork$ git stash list
stash@{0}: On develop: temporary changes 03
stash@{1}: On develop: temporary changes 02
stash@{2}: On develop: temporary changes 01
```

  - **Comando para aplicar a mudança salva na stash**
    - `$ git stash apply / $ git stash apply stash@{0}`

Referências

1. Diferenças entre ^ e ~: https://stackoverflow.com/questions/2221658/whats-the-difference-between-head-and-head-in-git
2. Diferenças entre o git reset --soft, --mixed, --hard: https://medium.com/@andgomes/os-tr%C3%AAs-tipos-de-reset-aa220658d9b2
3. Criando e configurando upstream para atualização de repositórios clonados via fork: https://gist.github.com/rdeavila/9618969
4. Guia rápido (em português): https://rogerdudler.github.io/git-guide/index.pt_BR.html
5. Simulador de branches: https://learngitbranching.js.org/?NODEMO
6. Criar chave ssh de acesso ao github: https://help.github.com/pt/github/authenticating-to-github/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

Obrigado pela atenção!
- Dúvidas?