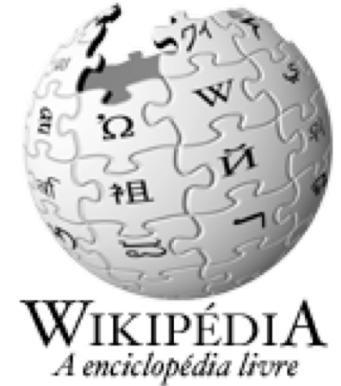


# IMD0030 LINGUAGEM DE PROGRAMAÇÃO I

Aula 09 – Controle de Versões (Git)

# Sistema de controle de versões



[https://pt.wikipedia.org/wiki/Sistema\\_de\\_controle\\_de\\_versões](https://pt.wikipedia.org/wiki/Sistema_de_controle_de_versões)

Um **sistema de controle de versões** (ou *versionamento*), **VCS** (do inglês *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* 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.

---

# Sistema de controle de versões

## Por que utilizar?

- Torna possível alterações feitas ao longo do desenvolvimento do *software*
- Permite ter um controle do histórico de todas as alterações feitas
  - Resgatar versões mais antigas e estáveis
  - Desfazer alterações que causem problemas
  - Identificar que usuário foi responsável por uma determinada alteração
- Permite ramificar o projeto, dividindo-o em diferentes linhas de desenvolvimento que podem ser trabalhadas em paralelo e de forma independente
- É uma ferramenta essencial no desenvolvimento colaborativo, em que comumente os membros da equipe trabalham à distância sobre o mesmo conjunto de arquivos

---

# Sistema de controle de versões

## Vocabulário

Termo	Definição
Repositório ( <i>repository</i> )	Local onde ficam armazenadas todas as versões dos arquivos, desde a primeira à última
Cópia local ( <i>working copy</i> )	Cópia dos arquivos do repositório na máquina local do usuário e sobre a qual ele irá trabalhar
Atualização ( <i>update</i> )	Atualização da cópia local com a última versão disponível no repositório, provavelmente resultante de alterações feitas por outro desenvolvedor
<i>Download</i> ( <i>checkout</i> )	<i>Download</i> dos os arquivos disponíveis no repositório, quando não há cópia local
Conflito ( <i>conflict</i> )	Alteração simultânea de um mesmo arquivo por usuários diferentes

---

# Sistema de controle de versões

## Vocabulário

Termo	Definição
Efetivação ou submissão ( <i>commit</i> )	Envio das alterações feitas sobre a cópia local para o repositório
Ramificação ( <i>branch</i> )	Divisão independente da linha de desenvolvimento principal ( <i>master</i> )
Mesclagem ( <i>merge</i> )	Integração de uma ramificação à linha de desenvolvimento principal ( <i>master</i> ), consolidando as alterações independentes que foram feitas
Diferença ( <i>diff</i> )	Diferença entre duas versões de um mesmo arquivo
Resolução de conflito ( <i>conflict solve</i> )	Análise do que entrou em conflito e decisão de qual alteração fará parte da versão final no repositório

---

# Sistema de controle de versões

Soluções livres mais comuns utilizadas atualmente



**Bazaar**



mercurial

# Git

- O **Git** é um sistema de controle de versões simples, rápido, confiável, distribuído e altamente gerenciável, sendo muito popular e largamente utilizado no mundo todo
- Criado em 2005 por Linus Torvalds para auxiliar no desenvolvimento do núcleo (*kernel*) do sistema operacional Linux
- Website oficial: <https://git-scm.com/>



# Git

Empresas e projetos que fazem uso do **Git**

**Google**

**facebook**



**Microsoft**

**twitter**

**LinkedIn**



**NETFLIX**



**PostgreSQL**

**eclipse**

**GNOME™**

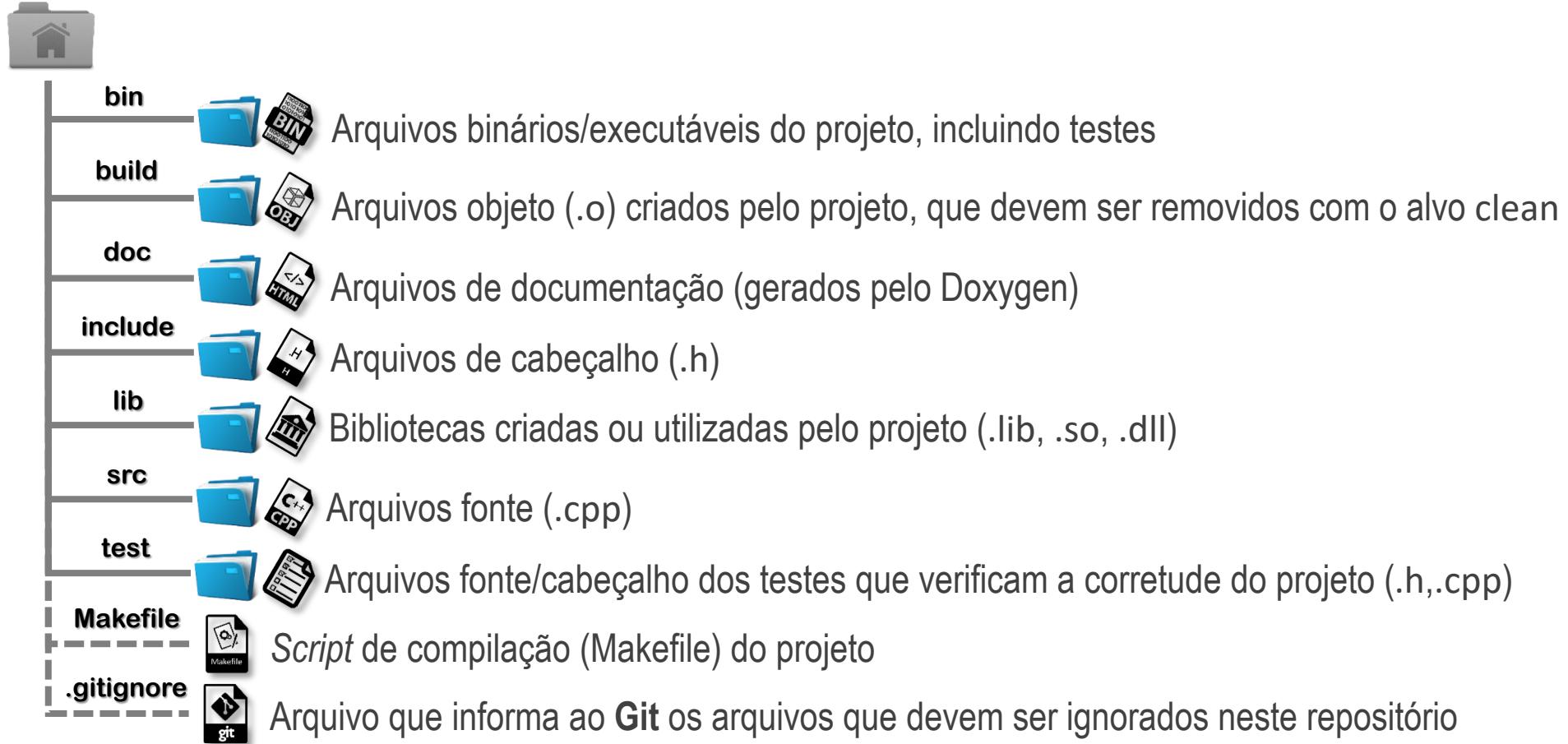


# Git

Serviços de hospedagem de repositórios que suportam o Git e que são comumente utilizados



# Estrutura básica de um repositório



---

# Git: Criando ou clonando repositórios

- Criando um novo repositório local
  - Crie um novo diretório e navegue para dentro dele (mkdir / cd)
  - Execute o comando git init para criar um novo repositório
    - Será criado um subdiretório oculto .git contendo as configurações do novo repositório criado
- Obtendo um repositório remoto já existente
  - Crie uma cópia local (*working copy*) de um repositório local executando o comando  
git clone <caminho para o repositório>
  - Se o repositório for remoto, o comando deve considerar as credenciais de acesso do usuário  
git clone <endereço do repositório remoto>  
Exemplo: git clone https://user@bitbucket.org/user/git-test.git

---

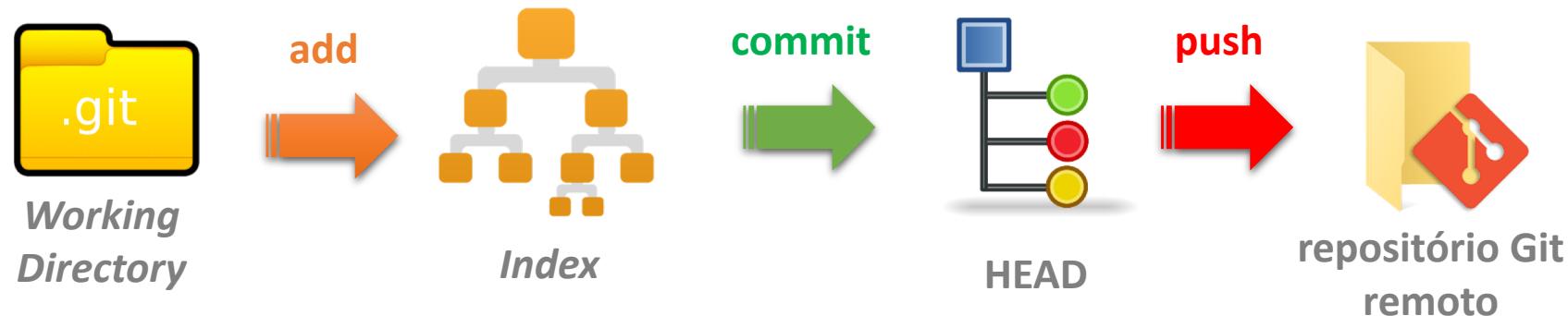
# Git: Ligando a um repositório remoto

- Caso você não tenha clonado um repositório existente, mas quer conectar o seu repositório a um servidor remoto, você pode adiciona-lo executando o comando `git remote add origin <endereço do servidor remoto>`
  - Exemplo: `git remote add origin https://user@bitbucket.org/user/git-test.git`
- Atualizando o seu repositório local com a mais nova versão
  - Execute o comando `git pull`

# Git: Estrutura do Repositório e Fluxo

## Fluxo de trabalho no Git

- O Git mantém três árvores (**trees**) em um repositório local:
  - **Working Directory**, que contém os arquivos sobre os quais o usuário está trabalhando
  - **Index**, que funciona como uma área temporária que controla aquilo que está sendo versionado
  - **HEAD**, que aponta para a última efetivação (**commit**) que foi realizada



---

# Git: Adicionar

- Propondo mudanças ao repositório para arquivos ainda não versionados, adicionando-os ao *index*
  - Execute o comando  
`git add <lista>` para adicionar uma lista de arquivos, pastas ou ambos  
`git add *` para adicionar todos os arquivos no diretório atual
- Efetivando as alterações (*commit*)
  - Execute o comando `git commit -m "Comentário descrevendo as alterações"`
    - Exemplo: `git commit -m "Corrigindo o bug #123"`
  - A execução do *commit* envia as alterações para o HEAD, mas ainda não para o repositório

---

# Git: Remover e Renomear

- Removendo um arquivo ou diretório do repositório local
  - Execute o comando:  
`git rm <lista_de_arquivo>` para remover um arquivo ou uma lista de arquivos  
`git rm -r <diretório>` para remover todo o conteúdo do diretório recursivamente
- Renomeando um arquivo
  - Execute o comando:  
`git mv <arquivo1> <arquivo2>`
- Efetive as alterações com um *commit*
  - Exemplo: `git commit -m "Arquivo teste.h removido."`
  - Exemplo: `git commit -m "Renomeando aquivo teste.h para cubo.h"`

# Git: O arquivo .gitignore

- Em um projeto há um conjunto de arquivos que não queremos ou precisamos inserir em um *commit* (e portanto não controlar versões)
  - Exemplos: arquivos temporários, arquivos de notas, etc.
- O arquivo `.gitignore` serve para ignorar arquivos não rastreados
  - A partir do momento em que um `git add` é utilizado para rastrear as mudanças do arquivo, o `.gitignore` não poderá ignorar este arquivo



```
1 # Ignora os diretorios build e lib
2 build lib/*
3
4 # Ignora todos os binarios/executaveis
5 bin/*
6
7 # Ignora arquivos especificos do Mac
8 .DS_Store
9
10 # Ignora o diretorio de documentacao
11 doc/*
12
```

---

# Git: Enviando as alterações

- Enviando as alterações para o repositório
  - Para enviar as alterações efetivadas no HEAD para o repositório, execute o comando `git push origin master`
    - As alterações são enviadas para a linha de desenvolvimento principal (*master*), mas é possível enviar-las para qualquer ramo (*branch*)
- Criando ramificações (*branches*)
  - É possível criar ramificações (*branches*) a partir da linha de desenvolvimento principal (*master*) ou de outros ramos através do comando `git checkout -b <nome para o branch>`
    - Para remover um *branch*, execute o comando `git branch -d <nome do branch>`
    - Um *branch* só estará disponível no repositório após a execução de um comando `git push` sobre ele

---

# Git: Gerenciando as alterações de múltiplos usuários

- Unificando alterações (*merge*)
  - Os ramos devem ser unificados após a conclusão das alterações
  - Para fazer *merge* de um *branch* ao seu *branch* ativo (*master*, por exemplo), execute o comando  
git merge <nome do branch>
    - O ideal é atualizar o *branch* ativo antes da realização do *merge*, executando o comando git pull
- Gerenciando conflitos no *merge*
  - Por padrão, o Git tenta realizar o *merge* das alterações automaticamente, porém isso nem sempre é possível devido a conflitos (diferenças entre arquivos que não podem ser resolvidas automaticamente)
  - Em caso de conflitos, é necessário resolve-los manualmente, editando os arquivos identificados pelo Git como conflitantes
  - Resolvido o conflito, é necessário adicionar os arquivos novamente por meio do comando git add

---

# Git: Gerenciando as alterações

- Para visualização do histórico de *commits*, execute o comando `git log`
- Para sobreescriver as alterações locais, execute o comando `git checkout --<arquivo>`
  - Este comando substitui as alterações na árvore de trabalho com o conteúdo mais recente no HEAD
  - Alterações já adicionadas ao *index*, bem como novos arquivos, são mantidos
- Para remover todas as alterações e *commits* locais:
  - Recupere o histórico mais recente do servidor executando o comando `git fetch origin`
  - Em seguida, aponte para o branch *master* local executando o comando  
`git reset --hard origin/master`

---

# Cientes Git

- Clientes de linha de comando
  - Mac OS: <https://code.google.com/archive/p/git-osx-installer/downloads>
  - Windows: <https://git-for-windows.github.io/>
  - Linux: <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>
- **Clientes gráficos** podem auxiliar na execução dos comandos Git, tornando a maioria deles transparente para o usuário
  - O link <https://git.wiki.kernel.org/index.php/InterfacesFrontendsAndTools> lista alguns clientes gráficos para Git disponíveis para diversos sistemas operacionais
- Ferramentas auxiliares para resolução de conflitos
  - SmartGit: <http://www.syntevo.com/smartgit/>
  - Atom merge-conflicts: <https://atom.io/packages/merge-conflicts>

# Alguma Questão?

