

# IMD0030

# LINGUAGEM DE PROGRAMAÇÃO I

Aula 04 – Controle de Versões e Documentação

# Parte I

## Controle de versões com 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 (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



---

# Git

Serviços hospedagem de repositórios utilizando o Git comumente utilizados

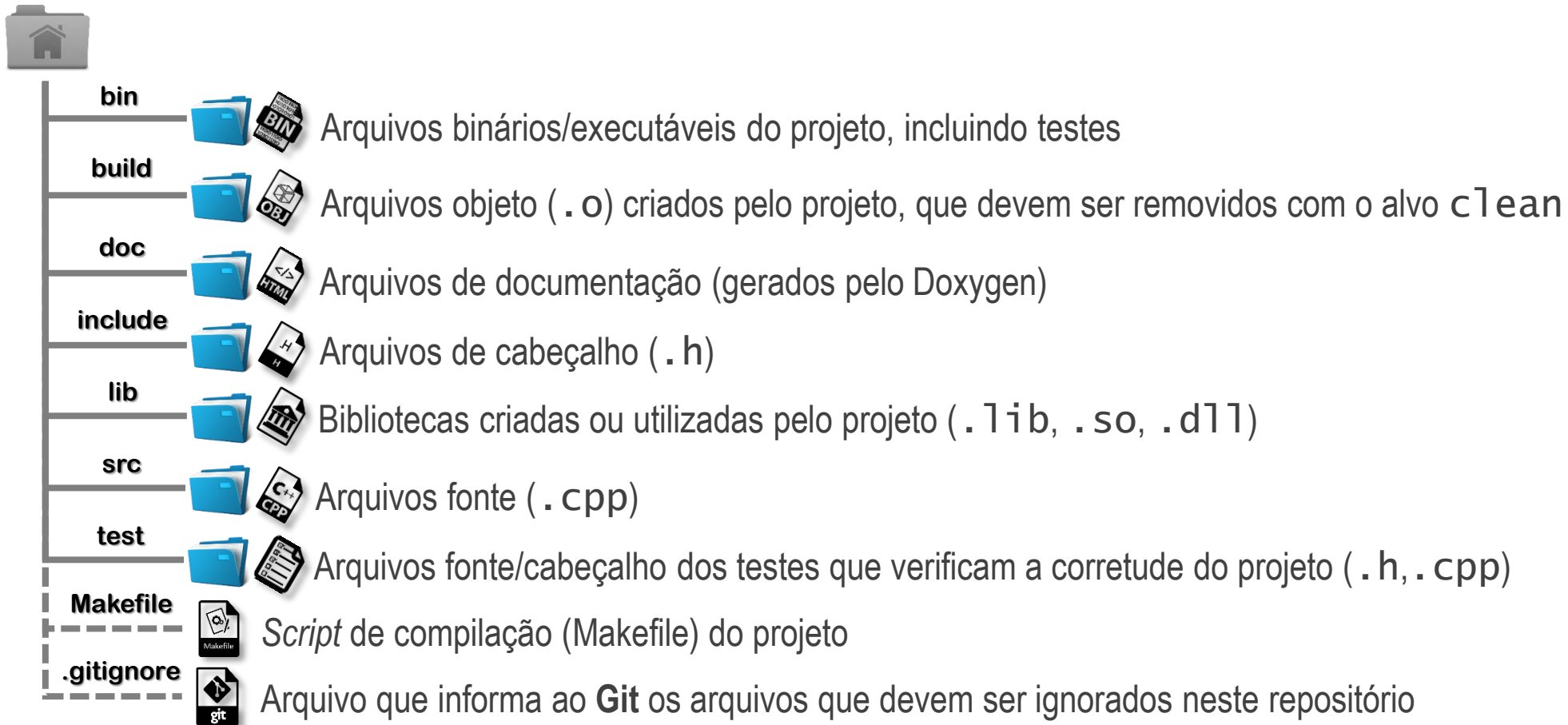


---

# Usando o Git (1)

- 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`

# Estrutura básica de um repositório



---

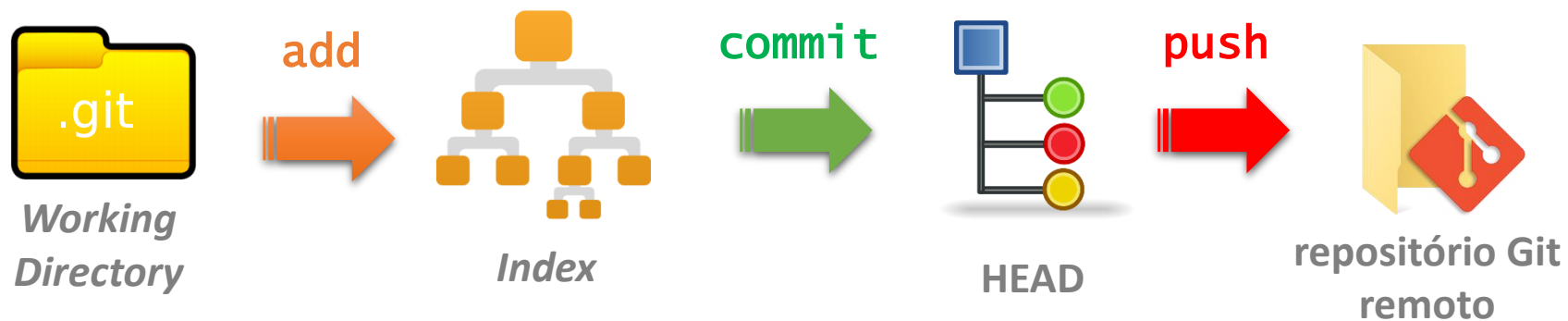
## Usando o Git (2)

- Caso você não tenha clonado um repositório existente, mas quer conectar o seu repositório a um servidor remoto, você pode adicioná-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`

# Usando o Git (3)

## 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



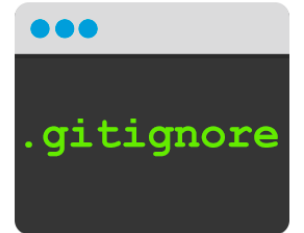
---

# Usando o Git (4)

- Propondo mudanças ao repositório para arquivos ainda não versionados, adicionando-os ao *index*
  - Execute o comando  
`git add <arquivo>` para adicionar um arquivo individual  
`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

# 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
```



---

# Usando o Git (5)

- 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 envia-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

---

# Usando o Git (6)

- 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 resolvê-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`

---

## Usando o Git (7)

- Para visualização do histórico de *commits*, execute o comando `git log`
- Para sobrescrever 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

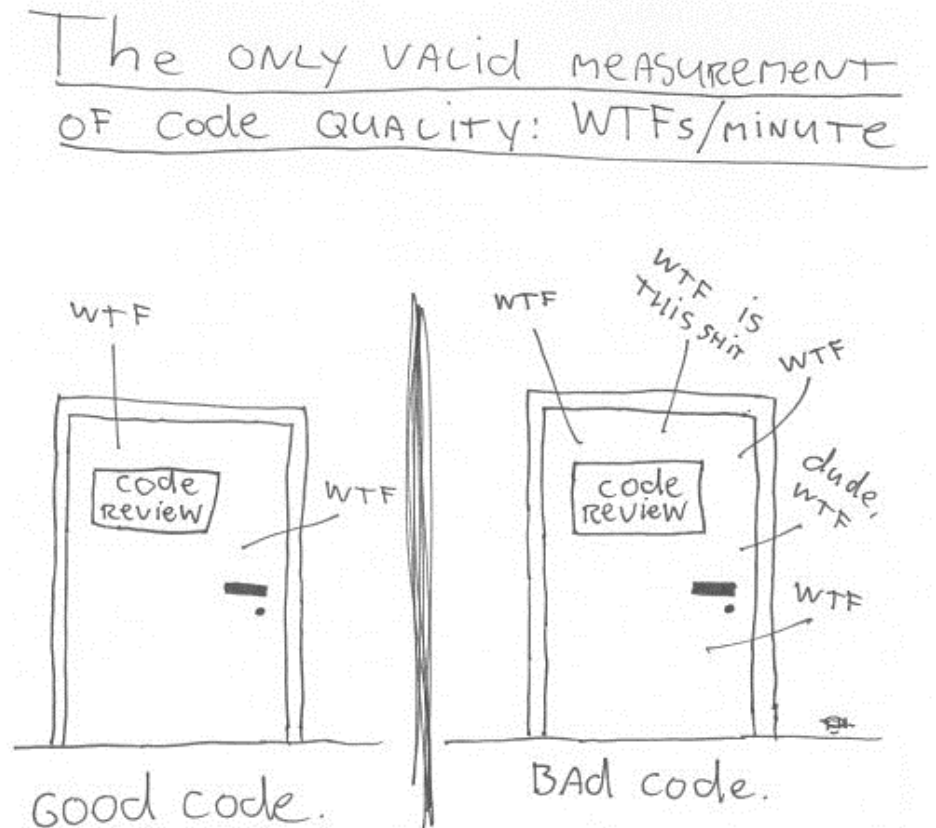
- Clientes de linha de comando
  - Mac OS: <https://code.google.com/archive/p/git-osx-installer/downloads>
  - Windows: <https://git-for-windows.github.io/>
  - Linux: <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>
- **Clientes gráficos** podem auxiliar na execução dos comandos Git, 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>

# Parte II

Documentação de código fonte com Doxygen

# A importância de boas práticas em programação

- Um código fonte, ainda que esteja ruim, **pode funcionar**
- Perdem-se **horas incontáveis e recursos importantes** por conta de um código mal escrito (mas que muitas vezes funciona)
- Um dos mecanismos para aumento na qualidade, compreensão, manutenção e evolução de código fonte por meio de **documentação**
  - A principal forma de documentação é por meio de **comentários**



---

# Comentários

## Bons comentários

- Comentários sucintos e que expliquem o porquê de alguma coisa no código
- Pendências a serem futuramente implementadas (TODOs) ou corrigidas (FIXMEs)
- Ênfase a alguma coisa que não pode passar despercebida
- Melhor explanação do funcionamento de alguma instrução, função, etc. cuja compreensão não seja imediata por um ser humano
  - Explicação de decisões de implementação
- Explicação de precondições, restrições e limitações



---

# Comentários

## Maus comentários

- Comentários com erros de escrita (ortografia e/ou gramática)
  - Comentários são lidos apenas por seres humanos e completamente ignorados por computadores
- Comentar tudo, incluindo aquilo que é óbvio demais (redundância)
  - Mais comentários que código em si é altamente prejudicial
- Comentários muito extensos, verbosos
- Comentários que não têm a ver com o que o código faz
  - Atentar para o caso de comentários tornarem-se obsoletos com relação ao código





---

# Geração automática de documentação

- Produzir documentação de qualidade, ao mesmo tempo que é fundamental no (re)uso de *software*, pode ser **uma tarefa relativamente difícil e demandar certo tempo**
- **Ferramentas para geração automática de documentação** podem auxiliar o desenvolvedor na tarefa de produzir uma boa documentação de *software*
  - Permitem que o desenvolvedor se concentre na elaboração da documentação do código (conteúdo), ao invés da aparência final de tal documentação
  - Mecanismos de **marcações** (ou **anotações**) no código fonte são utilizados pela ferramenta para gerar e organizar automaticamente a documentação final

---

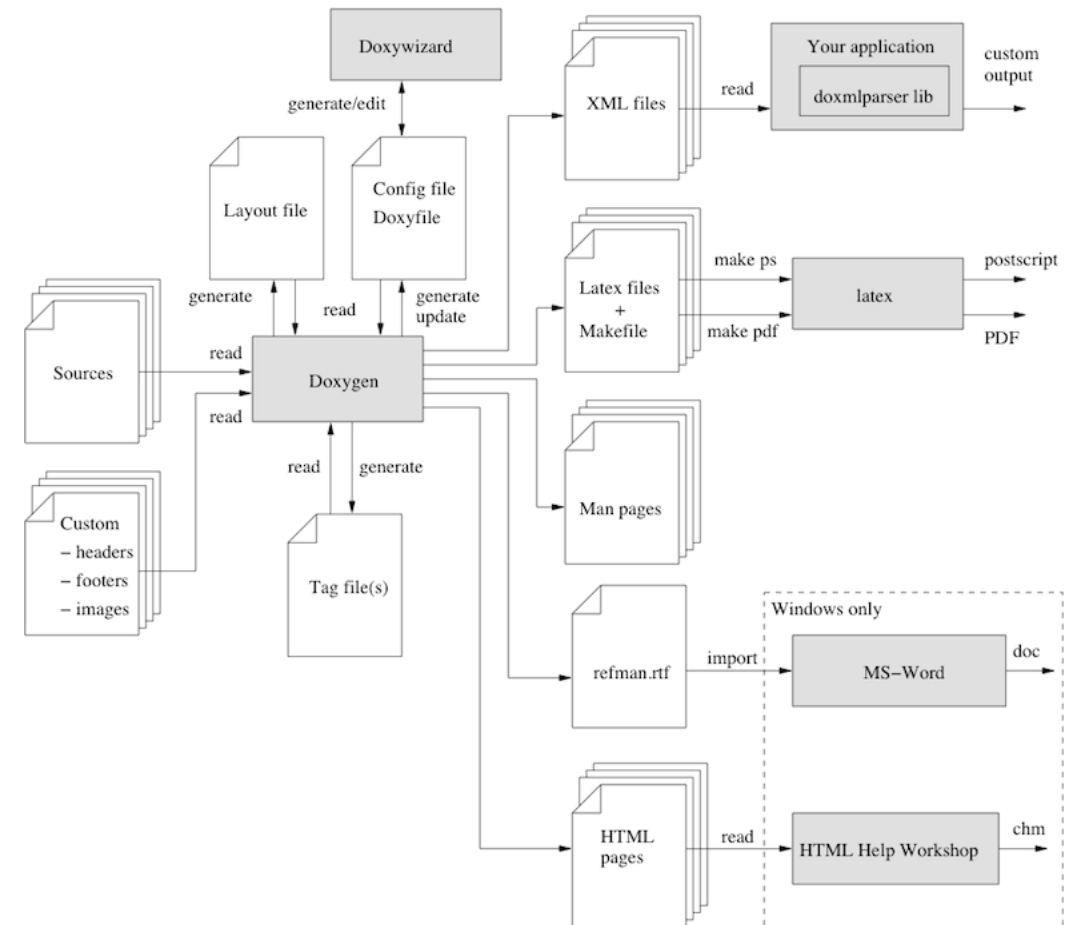
# Doxygen



- Ferramenta para geração automática de documentação de programas implementados nas linguagens de programação C, C++, Java, Objective-C, Python, PHP, etc.
- Disponível para os sistemas operacionais Windows, Linux e Mac OS
- Disponível em: <http://www.doxygen.org/>

# Como o Doxygen funciona

- Entrada:
  - **Arquivo de configuração (*Doxyfile*)** com um conjunto de opções que especificam alguns parâmetros referentes ao projeto em questão e como o Doxygen irá se comportar
  - **Arquivos de código fonte** comentados seguindo o estilo reconhecido pelo Doxygen
  - **Arquivos adicionais** (opcionais) para maior customização
- Saída: documentação na forma de páginas Web (HTML), LaTeX, RTF, XML e *man pages* para Linux



---

# Anotando o código fonte (1)

- Um bloco de documentação no estilo Doxygen **difere ligeiramente** do padrão de comentário existente nas linguagens de programação C e C++, pela adição de **marcadores**
- Os marcadores (anotações) permitem ao Doxygen reconhecer que aquela parte do arquivo deve ser utilizada no momento em que a documentação é gerada
- Relação completa de marcadores:  
<https://www.stack.nl/~dimitri/doxygen/manual/commands.html>
- O estilo de documentação mais utilizado para Doxygen é o **JavaDoc**:

```
/**  
 * ... texto ...  
 */
```

---

# Anotando o código fonte (2)

Possibilidades mais usadas para documentação com Doxygen

- **Descrição breve:** comentário de única linha (anotação `@brief`)
- **Descrição detalhada:** comentário que pode se estender por várias linhas (anotação `@details`)

```
/**  
 * @brief      Descricao breve  
 * @details    Descricao detalhada.  
 */
```

---

# Anotando o código fonte (3)

Possibilidades mais usadas para documentação com Doxygen

- **Documentação de membros** de estruturas, uniões, classes ou enumerações:
  - Uso do símbolo < em um bloco de comentário logo após a declaração do membro, **ou**
  - Inclusão de uma descrição breve (anotação `@brief`) antes da declaração do membro

```
int qtAlunos;      /**< Define a quantidade de alunos. O valor maximo  
e... */  
  
/** @brief Define a quantidade de alunos. O valor maximo e... */  
int qtAlunos;
```

---

# Anotando o código fonte (4)

Possibilidades mais usadas para documentação com Doxygen

- **Documentação de funções** em termos de descrições breves e/ou detalhadas, além da de seus parâmetros de entrada e retorno
  - Documentação de parâmetro de entrada: anotação `@param` seguida do nome do parâmetro
  - Documentação de retorno: anotação `@return`

```
/**  
 * @brief Funcao que calcula o fatorial de um numero  
 * @param n Numero cujo fatorial sera calculado  
 * @return Fatorial do numero  
 */  
long double fatorial(long double n);
```

---

# Anotando o código fonte (5)

Documentação do arquivo fonte propriamente dito

Marcador (anotação)	Descrição
@file	Documentação do arquivo fonte em questão
@author	Inserção do(s) nome(s) do(s) autor(es) do código fonte em questão. É possível separar os nomes de múltiplos autores por meio de vírgulas ou múltiplas anotações @author contendo o nome de um autor.
@since	Inserção da data de início da implementação
@date	Inserção de data (por exemplo, a data da última modificação do arquivo)
@version	Indicação da versão atual do arquivo
@sa	Inserção de referências cruzadas para classes, funções, métodos, variáveis, arquivos ou mesmo endereços da Internet



---

# Anotando o código fonte (6)

Documentação do arquivo fonte propriamente dito

```
/**  
 * @file      hello.cpp  
 * @brief     Primeiro programa na linguagem de programacao  
C++  
 * @author    Joao dos Anzois  
 * @since     01/01/2017  
 * @date      01/03/2017  
 * @sa        http://www.google.com/  
 */
```

---

# Gerando automaticamente a documentação

- Tendo-se o Doxygen devidamente instalado, o primeiro passo para a geração automática da documentação é criar o arquivo de configuração
  - No Linux, o Doxygen permite criar automaticamente o Doxyfile executando-se o comando `doxygen -g <nome do arquivo>`
  - Caso o nome do arquivo não seja fornecido, será criado um arquivo com o nome `Doxyfile`
  - O Doxyfile criado já contém uma série de opções (*tags*) estabelecidas por padrão, fazendo com que sejam necessárias pouquíssimas modificações
  - Relação completa de *tags*: <https://www.stack.nl/~dimitri/doxygen/manual/config.html>
- No Linux, a documentação é gerada automaticamente executando-se o comando `doxygen <arquivo de configuração>`
  - Caso o nome do arquivo tenha sido mantido com o padrão `Doxyfile`, basta executar `doxygen`

# Gerando automaticamente a documentação

Exemplo de documentação gerada automaticamente com o Doxygen

## IMC

Cálculo do índice de massa corporal (IMC) de uma pessoa e classificação do seu grau de obesidade

[Página Principal](#) [Arquivos](#)

**IMC Documentação**

Gerado por  1.8.11

## IMC


Cálculo do índice de massa corporal (IMC) de uma pessoa e classificação do seu grau de obesidade

[Página Principal](#) [Arquivos](#)

[Lista de Arquivos](#) [Membros dos Arquivos](#)

**Lista de Arquivos**

Esta é a lista de todos os arquivos documentados e suas respectivas descrições:

 <b>imc.cpp</b>	Programa que calcula o índice de massa corporea (IMC) de uma pessoa
--	---

Gerado por  1.8.11

# Gerando automaticamente a documentação

Exemplo de documentação gerada automaticamente com o Doxygen

## IMC

Cálculo do índice de massa corporal (IMC) de uma pessoa e classificação do seu grau de obesidade

Página Principal	Arquivos	Busca
Lista de Arquivos	Membros dos Arquivos	

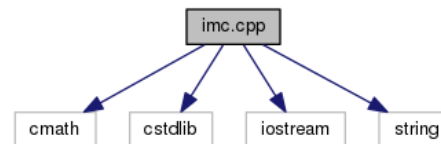
### Referência do Arquivo imc.cpp

Funções

Programa que calcula o índice de massa corporea (IMC) de uma pessoa. Mais...

```
#include <cmath>
#include <cstdlib>
#include <iostream>
#include <string>
```

Gráfico de dependência de inclusões para imc.cpp:



---

# Gerando automaticamente a documentação

Exemplo de documentação gerada automaticamente com o Doxygen

## Funções

float	<b>imc</b> (float peso, float altura)	Funcao que calcula o IMC de um individuo adulto a partir do seu peso e de sua altura. Mais...
string	<b>grau_obesidade</b> (float imc)	Funcao que determina o grau de obesidade de um individuo com base em seu IMC. Mais...
int	<b>main</b> (int argc, char *argv[])	Funcao principal. Mais...

## Descrição Detalhada

Programa que calcula o indice de massa corporea (IMC) de uma pessoa.

O IMC e uma medida internacional usada para calcular se uma pessoa esta no peso ideal. Essa medida e determinada pela divisão do peso da pessoa (em quilogramas) pelo quadrado de sua altura (em metros)

### Autor

Pedro Paulo Pereira

### Desde

01/01/2016

### Data

01/02/2016

---

# Gerando automaticamente a documentação

Exemplo de documentação gerada automaticamente com o Doxygen

```
float imc ( float peso,  
           float altura  
           )
```

Funcao que calcula o IMC de um individuo adulto a partir do seu peso e de sua altura.

#### Parâmetros

**Peso** em quilogramas

**Altura** em metros

#### Retorna

IMC do individuo

Gerado por [doxygen](#) 1.8.11

---

## Take away message

“Any fool can write code that a computer can understand.  
Good programmers write code that  
humans can understand.”



Martin Fowler

# Alguma Questão?

