

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Instituto de Informática
Departamento de Informática Aplicada

Aula 19: Projeto de software visando o reuso e controle de versionamento

Prof. Dennis Giovani Balreira
(Material adaptado dos professores Marcelo Pimenta e Ingrid Nunes)



INF01120 - Técnicas de Construção de Programas - 2022/1



Na aula passada...

1. Identificadores
2. Uso de variáveis
3. Tipos de dados
4. Codificação

- Outras linguagens orientadas a objetos
 - Top 5 linguagens mais utilizadas:
 - **Java**: POO puro, multiplataforma, software empresarial
 - **Python**: script de alto nível, fácil e intuitiva, foco na legibilidade
 - **C++**: nível intermediário, procedural e POO, com foco no desempenho
 - **C#**: multiparadigma, de uso geral usado a partir do framework .NET
 - **Ruby**: interpretada, multiparadigma e de alto nível, focada na simplicidade
 - Menções honrosas:
 - Javascript, Go, Swift, SQL, R, PHP
 - Objective-C, PHP, Scala, Kotlin, Rust, Visual Basic .NET

Projeto de software visando o reuso

Questões gerais sobre o reuso

- O que é artefato de software reusável?
- O que se espera da reusabilidade?
- Diferentes pontos de vista:
 - Cliente vs. provedor
- Como construir software reusável?
- Qual relação da reusabilidade com qualidade de software?
- Quais dificuldades para reusar software?

Don't Reinvent The Wheel !



Reuso de software

- Na maioria dos sistemas de engenharia, sistemas são projetados pela **composição de componentes** existentes **já usados** em outros sistemas
- Engenharia de software costuma focar no desenvolvimento **original**
 - Para obter software **mais rápido** e com **menor custo** é interessante adotar um processo de projeto baseado em **reuso de software**
- Problemas?



Reuso de software

- Na maioria dos sistemas de engenharia, sistemas são projetados pela **composição de componentes** existentes **já usados** em outros sistemas
- Engenharia de software costuma focar no desenvolvimento **original**
 - Para obter software **mais rápido** e com **menor custo** é interessante adotar um processo de projeto baseado em **reuso de software**
- Problemas?
 - Natureza humana é inerentemente **preguiçosa**
 - Desenvolvedores ficam felizes de encontrar **soluções prontas**
 - Uso **ad-hoc** (copy-paste) pode ser prejudicial!



Reuso de software - Perspectivas

- O reuso de software pode ser visto em seis perspectivas:
 - 1. Por substância
 - 2. Por escopo
 - 3. Por modo
 - 4. Por técnica
 - 5. Por intenção
 - 6. Por produto

Reuso de software - Perspectivas

- 1. Por substância
 - Define a **essência** dos itens a serem reusados
 - Reuso de conceitos abstratos (compartilhamento de conhecimento)
 - Ex: soluções gerais para classes de problemas (algoritmos)
 - Reuso de componentes
 - Ex: reuso de partes de software
 - Reuso de processos, procedimentos
 - Ex: uso informal de habilidades

Reuso de software - Perspectivas

- 2. Por escopo
 - Define a **forma** e **extensão** do reuso
 - Vertical
 - Reuso dentro do **mesmo domínio** ou área de aplicação
 - Ex: subrotinas para análise de DNA em biologia computacional
 - Horizontal
 - Reuso de partes **genéricas** em **diferentes aplicações**
 - Ex: subrotinas para análise de dados

Reuso de software - Perspectivas

- 3. Por modo
 - Define como o reuso é conduzido
 - Planejado (sistemático)
 - Software construído pensando no reuso
 - Ex: construir biblioteca matemática visando ser reusada
 - Ad-hoc (oportunista)
 - Software pensado no reuso após construção do mesmo
 - Ex: reusar funções matemáticas feitas em diferentes contextos

Reuso de software - Perspectivas

- 4. Por técnica
 - Define a **abordagem** para **implementar** o reuso
 - Composicional
 - Usa componentes existentes para construir novos sistemas
 - Ex: construir um editor de textos com GUI, bibliotecas IO, etc.
 - Generativa
 - Reuso através de geradores de aplicação (traduzem especificações para programas) *
 - Ex: gerador de tabelas em latex

* <http://www.linhadecodigo.com.br/artigo/476/reuso-em-nivel-de-geradores-de-aplicacao-como-elemento-de-ganho-de-productividade-em-programacao-orientada-a-objeto.aspx>

Reuso de software - Perspectivas

- 5. Por intenção
 - Define **como** os elementos serão reusados
 - Black-box (as-is)
 - Utilização do componente sem modificação
 - Ex: usar uma biblioteca de jogos sem modificações
 - White-box (modificada)
 - Utilização do componente com modificação para o adaptar às necessidades específicas
 - Ex: adaptar uma biblioteca de jogos para suportar física

Reuso de software - Perspectivas

- 6. Por produto
 - Define quais produtos serão reusados
 - Código fonte
 - Projeto
 - Especificações
 - Objetos
 - Texto
 - Arquiteturas

Reuso de software - Blocos de construção

- Tipos de **blocos** de reuso:
 - Reuso de **sistemas de aplicação**:
 - O **todo** de um sistema de aplicação
 - Ex: COTS*, Linhas de produto de software
 - Reuso de **componentes**
 - Ex: Subsistemas
 - Reuso de **objeto** e **função**:
 - Componentes de software que implementam um único conceito
 - Ex: Classe ArrayList (Java), Classe Vector (C++)

* COTS (Commercial Off-The Shelf): software de terceiros, comercial

Reuso de software - Características

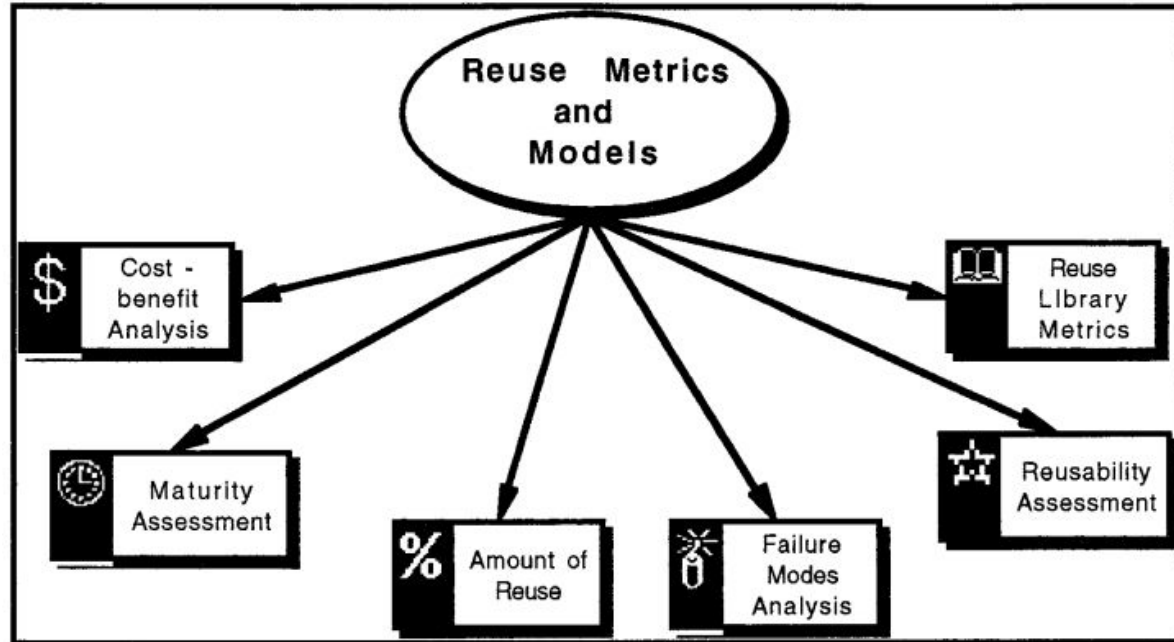
- **Benefícios:**

- Confiabilidade crescente
- Desenvolvimento com risco reduzido
- Uso efetivo de especialistas
- Conformidade com padrões
- Desenvolvimento acelerado

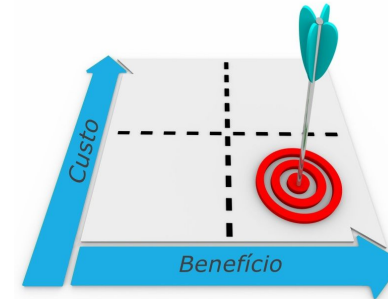
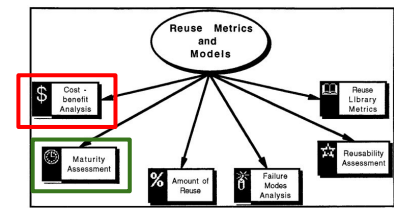
- **Problemas:**

- Custos de manutenção
- Falta de suporte ferramental
- Psicológicos (síndrome do *Not-Built-Here*)
- Criação e manutenção de componentes reusáveis (desenvolvedor)
- Buscar, entender e adaptar componentes reusáveis (cliente)

Reuso de software - Modelos e métricas



Reuso de software - Modelos e métricas

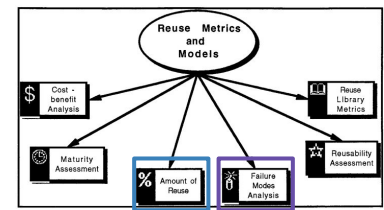


- **Análise do custo-benefício:**
 - Justificar o **custo** e **tempo** investido em reuso, estimando o potencial retorno financeiro

- **Avaliação de maturidade:**
 - Modelo de maturidade de reuso no qual avaliam o quão avançada a **organização** está na **implementação** de reuso de forma sistemática

	1 Initial/ Chaotic	2 Monitored	3 Coordinated	4 Planned	5 Ingrained
Motivation/ Culture	Reuse discouraged	Reuse encouraged	Reuse incentivized re-enforced rewarded	Reuse indoctrinated	Reuse is the way we do business
Planning for reuse	None	Grassroots activity	Targets of opportunity	Business imperative	Part of strategic plan
Technology support	Personal tools, if any	Many tools, but not specialized for reuse	Classification aids and synthesis aids	Electronic library separate from development environment	Automated support integrated with development environment
Metrics	No metrics on reuse level, pay-off, or costs	Number of lines of code used in cost models	Manual tracking of reuse occurrences of catalog parts	Analyses done to identify expected payoffs from developing reusable parts	All system utilities, software tools and accounting mechanisms instrumented to track reuse
Legal, contractual, accounting considerations	Inhibitor to getting started	Internal accounting scheme for sharing costs and allocating benefits	Data rights and compensation issues resolved with customer	Royalty scheme for all suppliers and customers	Software treated as key capital asset

Reuso de software - Modelos e métricas



- Quantidade de reuso:

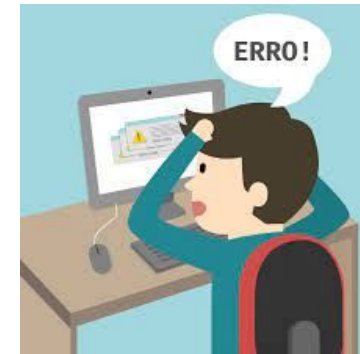
- Usadas para **avaliar** e **monitorar** o esforço de reutilização, acompanhando durante o ciclo de vida

$$\frac{\text{amount of life cycle object reused}}{\text{total size of life cycle object}}$$

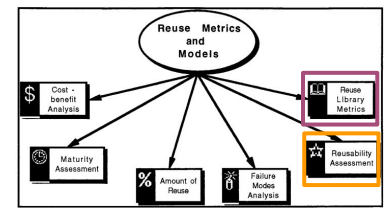
$$\frac{\text{lines of reused code in system or module}}{\text{total lines of code in system or module}}$$

- Modelo de forma de falhas:

- Provê uma forma de **melhorar** o programa de reuso baseado nas **formas** em que pode haver falha
 - Atua na melhoria do processo considerando formas em que o reuso pode falhar



Reuso de software - Modelos e métricas



- **Avaliação de reusabilidade:**

- Consiste em **estimar** a reusabilidade de um determinado componente de software
 - Taxa de comentários por linha de código
 - Quantidade de chamadas a outros módulos por linha de código

```
1 # Debugging simple variable statements.
2 number = 100
3 print(f'Number is {number}')
4
5 number = 500
6 print(f'New Number is {number}')
7
8
9 # Debugging a for loop
10 userList = ['Rob', 'Dave', 'Fred']
11
12 for user in userList:
13     print(user)
```

- **Métricas para bibliotecas de reuso:**

- Formas de **buscar** as **melhores** bibliotecas de reuso para otimizar a efetividade da sua utilização
 - Time in use
 - Reuse reviews
 - Testing
 - ...



Reuso de software - Visões

- Padrões de projeto
 - Abstrações genéricas para projetos
- Bibliotecas de programas
- Desenvolvimento baseado em componentes
 - Sistemas desenvolvidos pela ligação de componentes
- Frameworks de aplicação:
 - Coleção de classes que podem ser adaptadas e estendidas
- ...

Reuso de software - Planejamento

- Fatores de planejamento:
 - Cronograma de desenvolvimento
 - Vida útil esperada para o software
 - Conhecimentos prévios, habilidades, experiência da equipe de desenvolvimento
 - Requisitos do software (funcionais e não funcionais)
 - Domínio da aplicação
 - Plataforma de execução do software

Reuso de software - Princípios de projeto de pacote

- Princípios **gerais**:
 - **Agregar** classes **voláteis** (que se alteram), deixando-as juntas
 - Isolar classes que mudam com frequência
 - **Separar** classes que mudam por **razões distintas**
 - **Separar** preocupações de **alto nível** com as de **baixo nível**
 - Ex:
 - Alto nível: interação com o usuário
 - Baixo nível: leitura de arquivos

Reuso de software - Princípios de projeto de pacote

- Granularidade:
 - Reutilizar **pacotes** (e não classes individuais)
 - Gerenciamento de release: versões antigas, anunciar mudanças, etc.
 - Difícil de fazer isso por classe
 - Classes em pacote devem ser protegidas (**fechadas**) a **mudanças**
 - Agrupar classes por suscetibilidade a mudanças
 - Classes em um pacote devem ser reusadas **juntas**
 - Se espalhadas, mudanças afetarão múltiplos pacotes

Reuso de software - Princípios de projeto de pacote

- Estabilidade:
 - Ciclos no grafo de dependências de pacotes não devem ser permitidos
 - Forma “acoplamento” do pacote
 - O que é afetado quando o pacote do ciclo é modificado?
 - Pacote não deve depender de outros pacotes menos estáveis
 - São mais prováveis de mudar, modificando outros pacotes
 - Pacote deve ser tão abstrato quanto estável
 - Se o pacote for mais “abstrato” outros podem usá-lo sem mudá-lo

Reuso de software - Técnicas para reuso

- **Polimorfismos** dinâmico (sobreescreita) e estático (sobrecarga)
- **Genericidade** (templates)
 - Habilidade de definir módulos **parametrizados**
 - Possibilita a escrita do **mesmo código** para descrever o **mesmo conceito** aplicado a **diferentes tipos de dados**

- Módulo modelo
 - TABLE_HANDLING [G]
- Módulos reais são obtidos pelo fornecimento de parâmetros genéricos reais (módulos derivados)
 - TABLE_HANDLING [INTEGER]
 - TABLE_HANDLING [REAL]
 - TABLE_HANDLING [ACCOUNT]

Reuso de software - Técnicas para reuso

- Java Generics
 - Permite criar uma classe só e, a partir dessa classe, instanciar objetos de diferentes tipos, de acordo com a escolha do programador

`ArrayList<String> al = new ArrayList<String>();`

Base-type

Type-parameter T

Here, ArrayList is Base-type and String is Type-parameter

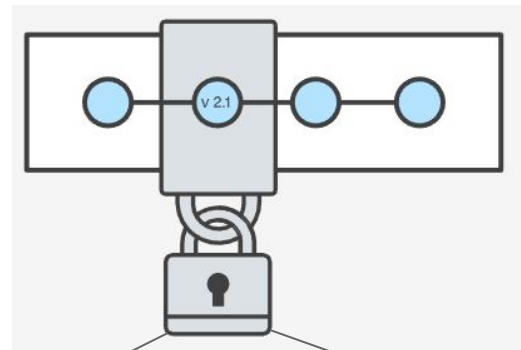
```
public class MinhaClasse<T> {  
    private T obj;  
  
    public MinhaClasse(T obj) {  
        this.obj = obj;  
    }  
  
    public void printar() {  
        System.out.println(obj);  
    }  
}  
  
// ----- Classe para testar  
public class MinhaClasseTeste {  
    public static void main(String[] args) throws Exception {  
        MinhaClasse<Integer> inteiro = new MinhaClasse<>(2);  
        inteiro.printar();  
        MinhaClasse<Double> decimal = new MinhaClasse<>(2.0);  
        decimal.printar();  
        MinhaClasse<String> palavra = new MinhaClasse<>("Sou Str");  
        palavra.printar();  
    }  
}
```

Mais detalhes em: <https://blog.cod3r.com.br/generics-em-java-o-que-e-e-exemplos/>

Controle de versionamento

Controle de versionamento

- Prática de **rastrear** e **gerenciar** as alterações em um código de software
- Sistemas de controle de versão permitem que as equipes de software **gerenciem** mudanças no projeto ao longo do tempo
- O sistema mantém **registro** de todas as modificações no código (e nos dados)
 - Se erro for cometido, desenvolvedores podem voltar no tempo e comparar versões anteriores do código



<https://www.atlassian.com/br/git/tutorials/what-is-version-control>

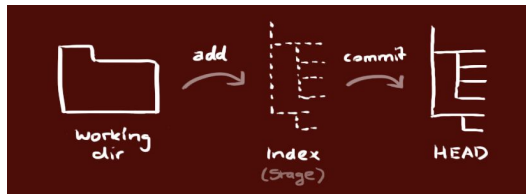
Controle de versionamento - Git

- Git é um sistema de controle de versões distribuído
- Instalação
 - OSX (<http://git-scm.com/download/mac>)
 - Windows (<http://msysgit.github.io/>)
 - Linux (http://book.git-scm.com/2_installing_git.html)
- Interfaces Gráficas para auxílio (opcional):
 - Gitk (<https://git-scm.com/docs/gitk/>)
 - TortoiseGit (<https://tortoisegit.org/>)
 - Obs: Algumas IDEs e editores já oferecem suporte embutido para o Git (ex: VsCode)

Controle de versionamento - Git: Guia Prático

- Obter um repositório:
 - Cria uma cópia de trabalho em um repositório local executando o comando:
- Fluxo de Trabalho: os repositórios locais consistem em três "árvores" do git
 - Working Directory que contém os arquivos vigentes.
 - Index que funciona como uma área temporária
 - HEAD que aponta para o último commit (confirmação) que você fez

```
git clone /caminho/para/repositorio
```



https://rogerdudler.github.io/git-guide/index.pt_BR.html

Controle de versionamento - Git: Guia Prático

- Adicionar e Confirmar:

- Você pode propor **mudanças** (adicioná-las ao Index) usando:

```
git add <arquivo>
```

```
git add *
```

- Este é o primeiro passo no fluxo de trabalho básico do git. Para realmente **confirmar** estas mudanças (isto é, fazer um commit), use

```
git commit -m "comentários das alterações"
```

- Agora o arquivo é enviado para o **HEAD**, mas ainda **não** para o repositório **remoto**.

https://rogerdudler.github.io/git-guide/index.pt_BR.html

Controle de versionamento - Git: Guia Prático

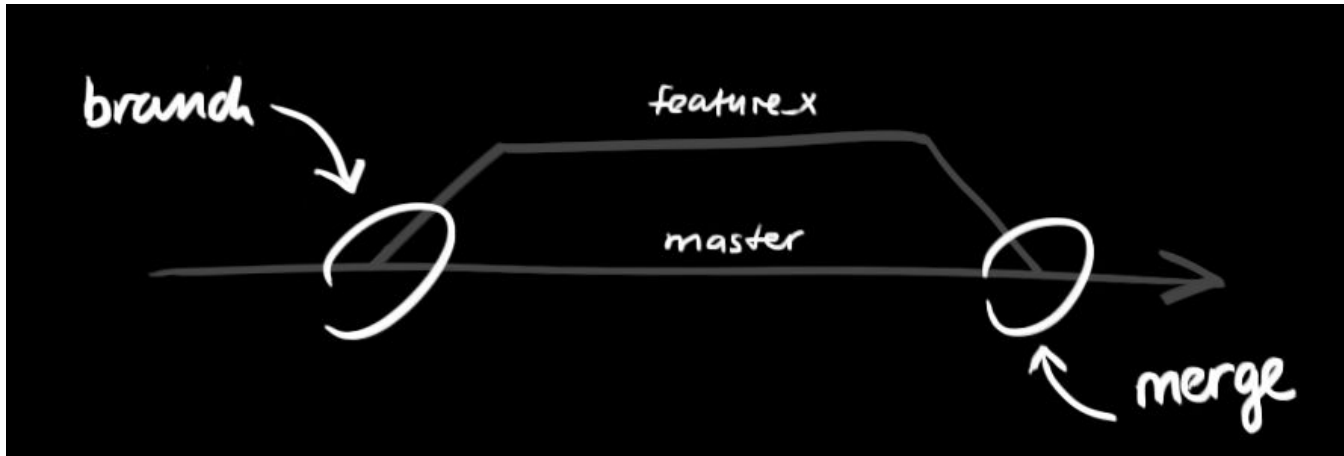
- Enviando alterações para o repositório remoto:
 - Suas alterações agora estão no **HEAD** da sua **cópia** de trabalho local. Para enviar estas alterações ao seu repositório remoto, execute

```
git push origin master
```
 - Altere master para qualquer ramo (*branch*) desejado, enviando suas alterações para ele.

https://rogerdudler.github.io/git-guide/index.pt_BR.html

Controle de versionamento - Git: Guia Prático

- **Branches** ("ramos") são utilizados para desenvolver funcionalidades isoladas umas das outras. O branch master é o branch "padrão" quando você cria um repositório. Use outros branches para desenvolver e mescle-os (**merge**) ao branch master após a conclusão.



https://rogerdudler.github.io/git-guide/index.pt_BR.html

Controle de versionamento - Git: Guia Prático

- Crie um novo *branch* chamado "funcionalidade_x" e selecione-o usando

```
git checkout -b funcionalidade_x
```

- Retorne para o *master* usando

```
git checkout master
```

- e remova o *branch* da seguinte forma

```
git branch -d funcionalidade_x
```

- Um *branch* não está disponível a outros a menos que você envie o *branch* para seu repositório remoto

```
git push origin <funcionalidade_x>
```

https://rogerdudler.github.io/git-guide/index.pt_BR.html

Controle de versionamento - Git: Guia Prático

- Atualizar e Mesclar:
 - Para atualizar seu repositório local com a mais nova versão, execute
`git pull`
na sua pasta de trabalho para obter e fazer merge (mesclar) alterações remotas.

https://rogerdudler.github.io/git-guide/index.pt_BR.html

Controle de versionamento

- Existem diversas **plataformas** de hospedagem de **código fonte** e **arquivos** com controle de versão usando Git
- Exemplos:
 - **GitHub**
 - **GitLab**
 - **BitBucket**

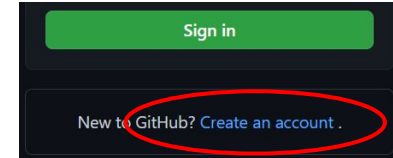


Controle de versionamento - Github: contexto histórico

- O GitHub foi desenvolvido por Chris Wanstrath, J. Hyett, Tom Preston-Werner e Scott Chacon usando [Ruby on Rails](#), e começou em fevereiro de 2008
- A empresa, GitHub, Inc., existe desde 2007 e está localizada em São Francisco
- Em 2018, Microsoft realiza a compra do GitHub por US\$ 7,5 bilhões

Controle de versionamento - Github: setup inicial

1. Crie uma conta no Github ([Sign up](#))



2. Configure o e-mail (use o mesmo utilizado no GitHub) e usuário do Git da sua máquina (terminal):

- `$ git config --global user.name "Your Name"`
- `$ git config --global user.email "your@email.com"`

3. Crie uma chave SSH

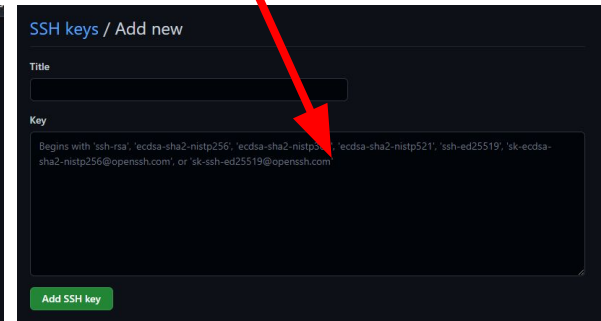
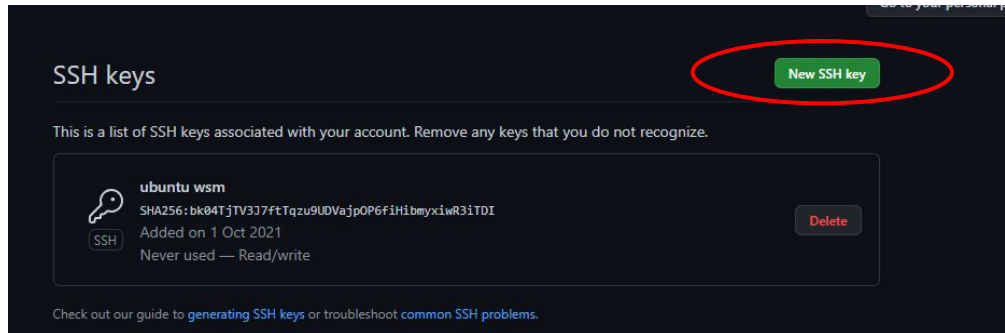
- `$ ssh-keygen -t rsa -C "your.email@example.com" -b 4096`
- Será solicitado o caminho onde salvar a chave (id_rsa.pub) e uma passphrase, **deixe ambos em branco e apenas dê ENTER.**

Controle de versionamento - Github: setup inicial

4. Copie o conteúdo do arquivo gerado ("~/.ssh/id_rsa.pub") com o seguinte comando:

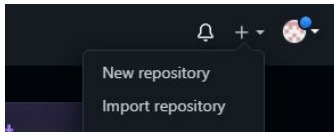
- `$ cat ~/.ssh/id_rsa.pub | clip`

5. Cole o conteúdo copiado na área reservada para ssh keys no seu perfil do GitHub (<https://github.com/settings/ssh/new>)

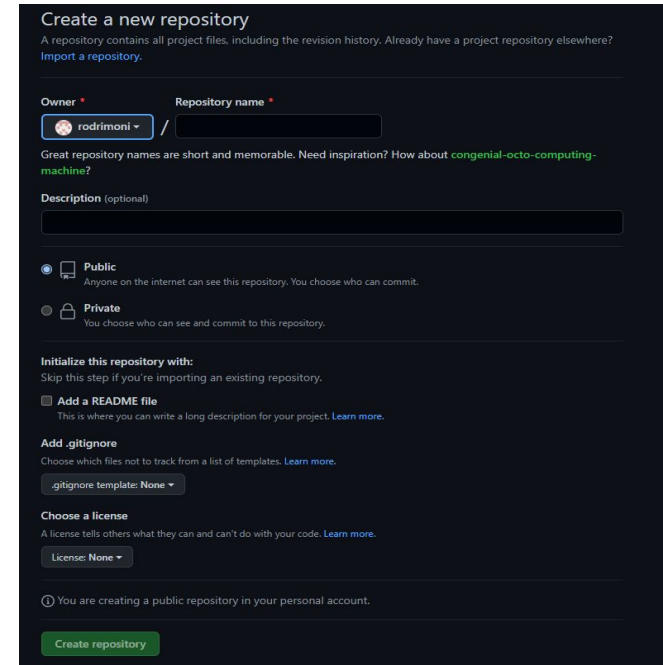


Controle de versionamento - Github: criação do repositório

1. No canto superior direito de qualquer página, use o menu drop-down e selecione New repository



2. Defina um nome para o repositório
3. Escreva uma breve descrição na caixa “description”
4. Selecione [Add a README file](#)
 - Arquivos README são escritos em texto utilizando text markdown language; É possível utilizar o seguinte código para saber mais sobre a sintaxe Markdown: [Markdown Cheat Sheet](#)
5. Selecione se seu repositório será Público ou Privado
6. Clique em [Create repository](#)

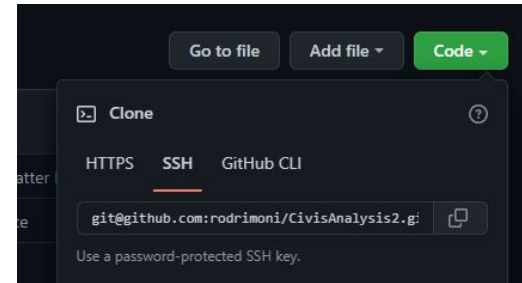
A screenshot of the 'Create a new repository' form on GitHub. The form is titled 'Create a new repository' and includes a subtitle 'A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.' The form fields include: 'Owner' (dropdown menu showing 'rodrimoni'), 'Repository name' (text input), 'Description (optional)' (text input), 'Public/Private' (radio buttons), 'Initialize this repository with:' (checkbox for 'Add a README file'), 'Add .gitignore' (checkbox), 'Choose a license' (dropdown menu), and a 'Create repository' button at the bottom.

Controle de versionamento - Github

- Agora você deverá clonar o seu repositório em sua máquina local, para isso abra a página do seu repositório recém criado e pegue o [/caminho/para/repositorio](#) clicando no botão **Code** e depois em **SSH**, logo após copie aquele endereço e execute o seguinte comando no terminal:

```
git clone /caminho/para/repositorio
```

- **Pronto!** Agora em você terá uma pasta com a cópia local do seu repositório e pode seguir para os exercícios.



Controle de versionamento - Github

- Exercício 1 - Considere o seguinte código abaixo em Python utilizando paradigma procedural:

```
nome = str(input('Digite seu nome: '))  
print('Ola ', nome, ' bem-vindo a TCP!')
```

1. Crie um arquivo “nome.py” (dentro da pasta do repositório clonado) que contenha este código
2. Faça o commit e o push inicial
 - Digite `git status` para verificar o que foi modificado
 - Digite `git add *` para adicionar suas mudanças
 - Digite `git commit -m “Meu primeiro commit!”` para confirmá-las
 - Digite `git push origin main`

Controle de versionamento - Github

- Exercício 2 - Considere um código simples em Python que implementa as quatro operações básicas de uma [calculadora](#), utilizando paradigma procedural, disponível no Moodle. Siga as seguintes instruções:
 1. Repita as instruções 1 e 2 do Exercício 1
 2. Crie uma nova branch a partir do seu branch principal
 3. Implemente uma nova função (localmente) para calcular a exponencial dada a base e o expoente como entrada
 4. Faça commit e push para enviar as alterações para o Github
 5. Faça um pull-request para informar outras pessoas sobre as alterações feitas no passo anterior

Controle de versionamento - Github

- Links para saber mais sobre o git:
 - [git - guia prático - sem complicação!](#) (muito bom para relembrar os principais comandos)
 - [Git Tutorial](#)

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Instituto de Informática
Departamento de Informática Aplicada

Obrigado pela atenção!
Dúvidas?

Prof. Dennis Giovani Balreira
(Material adaptado dos Profs. Marcelo Pimenta e Ingrid Nunes)



INF01120 - Técnicas de Construção de Programas - 2022/1

