

Monitoria em Ciência da Computação

Aula 04 - Versionamento, git & GitHub
Prof. Stefano Mozart



Criando um Ambiente Virtual Python





O comando `virtualenv`

- ❑ Na aula passada, realizamos a criação de um ambiente virtual Python com auxílio do Pycharm. Agora, usaremos o comando `virtualenv` para criar um ambiente virtual manualmente;

```
virtualenv [OPTIONS] DEST_DIR
```

- ❑ Após criar o ambiente virtual, é necessário ativá-lo, executando o script criado em `DEST_DIR/bin/activate` :
 - ❑ No windows: `DEST_DIR/bin/activate.bat` ;
 - ❑ No Linux: `source DEST_DIR/bin/activate` .



Versionamento





Versionamento

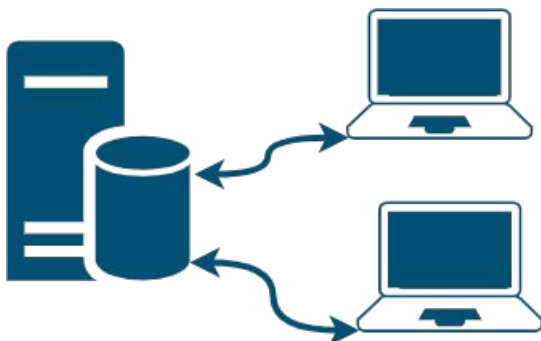
- ❑ Versionamento, ou “Controle de Versão” é a prática de registrar e gerir as mudanças em ativos digitais;
- ❑ É uma prática geralmente associada ao controle de mudança e configuração de software, mas também é amplamente utilizada em gestão de conteúdo textual, de imagens etc;
- ❑ O versionamento geralmente acontece com uso de um *software* de Controle de Versão, que permite o registro, visualização e alteração do histórico de alterações em um arquivo ou conjunto de arquivos;



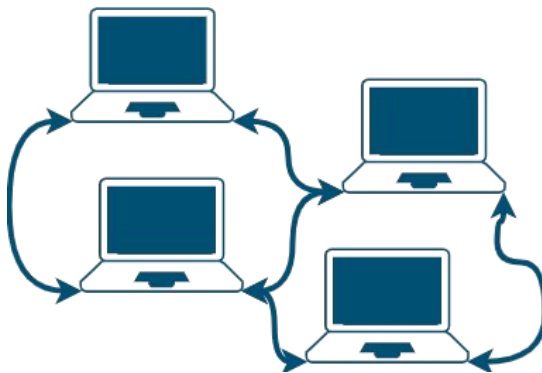
Versionamento

Existem vários tipos de sistemas controladores de versão, entre os quais podemos destacar:

Sistema centralizado de controle: o banco de dados guarda um histórico linear de alterações e é centralizado em um servidor

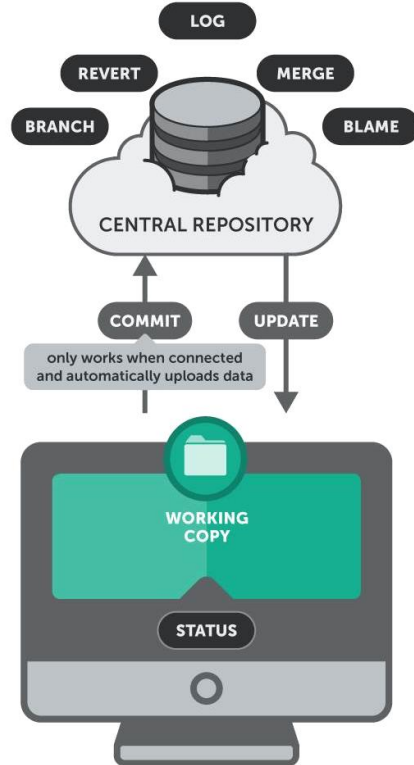


Sistema distribuído de controle: cada máquina guarda um histórico, na forma de grafo, que pode conter “porções” vindas de outras máquinas

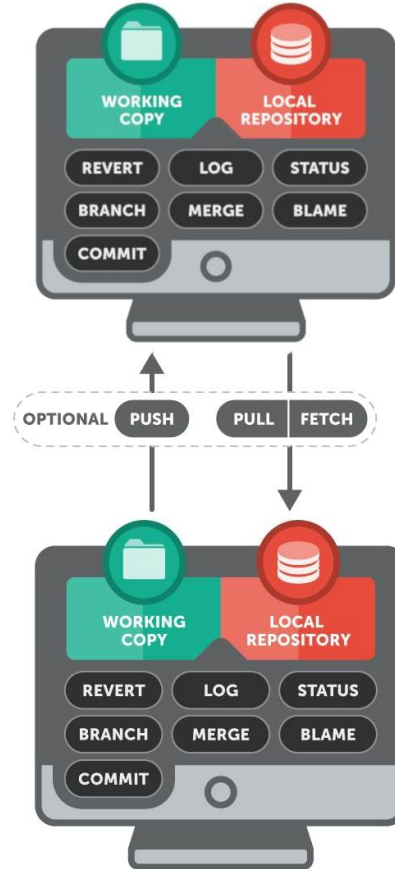


Versionamento

Centralizado



Descentralizado



Git





Git

- ❑ O termo Git pode referir-se a três coisas:
 - ❑ Uma arquitetura distribuída de controle de versão;
 - ❑ Um protocolo de transmissão de arquivos e metadados associados;
 - ❑ O software de controle de versão que implementa a arquitetura Git e, opcionalmente, o protocolo Git;
- ❑ A primeira implementação do versionador Git foi criada por Linus Torvalds, líder do desenvolvimento do kernel no Linux, em 2015;
- ❑ Atualmente, existem várias implementações do versionador e diversos serviços de Software-as-a-Service (e.g. Github, Gitlab, BitBucket) que seguem a arquitetura de versionamento do Git;

Git Cheat Sheet

Remember!
`git <COMMAND> --help`

Global configuration is stored in `~/.gitconfig`.
`git config --help`

master is the default development branch.
origin is the default upstream repository.

✦ Create

From existing data
`cd ~/my_project_directory`
`git init`
`git add .`

From existing repository
`git clone ~/existing_repo ~/new/repo`
`git clone git://host.org/project.git`
`git clone ssh://user@host.org/project.git`

✦ Show

Files changed in working directory
`git status`

Changes made to tracked files
`git diff`

What changed between ID1 and ID2
`git diff <ID1> <ID2>`

History of changes
`git log`

History of changes for file with diffs
`git log -p <FILE> <DIRECTORY>`

Who changed what and when in a file
`git blame <FILE>`

A commit identified by ID
`git show <ID>`

A specific file from a specific ID
`git show <ID>:<FILE>`

All local branches
`git branch`
star (*) marks the current branch

✦ Revert

Return to the last committed state
`git reset --hard`
This cannot be undone!

Revert the last commit
`git revert HEAD`
Creates a new commit

Revert specific commit
`git revert <ID>`
Creates a new commit

Fix the last commit
`git commit -a --amend`
(after editing the broken files)

Checkout the ID version of a file
`git checkout <ID> <FILE>`

✦ Update

Fetch latest changes from origin
`git fetch`
(this does not merge them)

Pull latest changes from origin
`git pull`
(does a fetch followed by a merge)

Apply a patch that someone sent you
`git am -3 patch.mbox`
In case of conflict, resolve the conflict and
`git am --resolved`

✦ Publish

Commit all your local changes
`git commit -a`

Prepare a patch for other developers
`git format-patch origin`

Push changes to origin
`git push`

Make a version or milestone
`git tag v1.0`

✦ Branch

Switch to a branch
`git checkout <BRANCH>`

Merge BRANCH1 into BRANCH2
`git checkout <BRANCH2>`
`git merge <BRANCH1>`

Create branch BRANCH based on HEAD
`git branch <BRANCH>`

Create branch BRANCH based on OTHER
and switch to it
`git checkout -b <BRANCH> <OTHER>`

Delete branch BRANCH
`git branch -d <BRANCH>`

✦ Resolve merge conflicts

View merge conflicts
`git diff`

View merge conflicts against base file
`git diff --base <FILE>`

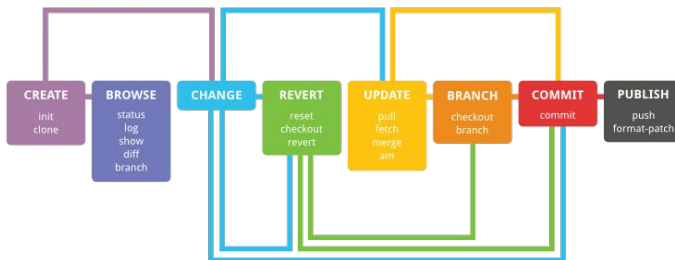
View merge conflicts against your changes
`git diff --ours <FILE>`

View merge conflicts against other changes
`git diff --theirs <FILE>`

Discard a conflicting patch
`git reset --hard`
`git rebase --skip`

After resolving conflicts, merge with
`git add <CONFLICTING_FILE>`
`git rebase --continue`

✦ Workflow



GitHub





GitHub

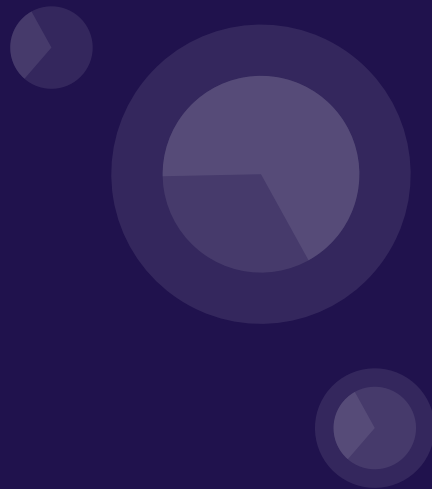
- ❑ GitHub é um serviço de controle de versão que, além de hospedar repositórios git, proporciona diversas funcionalidades de colaboração úteis para o gerenciamento de um projeto de software:
 - ❑ Projetos e organizações - mecanismos de agrupamento e controle de acesso, para gestão de repositórios;
 - ❑ Issues - registros de problemas ou pedido de melhoria;
 - ❑ Pull requests - pedidos para que os mantenedores do repositório aceitem uma alteração feita por terceiros;
 - ❑ Packages - mecanismos de distribuição de uma versão específica do código num repositório;



GitHub

- ❑ O GitHub também oferece diversas funcionalidades semelhantes às de redes sociais, tais como **star**, **watch** e **wikis** (para repositórios), ou **profile**, **follow**, **gist**, **pages** e **discussions** (para usuários);
- ❑ Uma funcionalidade de colaboração importantíssima no GitHub é a de **pull requests**, que são como uma requisição enviada aos mantenedores de um repositório para que aceitem um pacote de mudanças:
 - ❑ Os mantenedores podem aceitar o **pull request** assim como foi enviado, acatando as alterações propostas;
 - ❑ Os mantenedores também podem usar o **pull request** como um *branch*, fazendo ajustes até que aquele pacote de mudanças possa ser incorporado ao repositório;

Git-Bash





Git-Bash

- ❑ Git-Bash é uma aplicação para sistemas Windows que inclui um emulador de terminal de comandos, uma implementação do versionar Git e também alguns utilitários (ls, touch, mkdir, rm, entre outros);
- ❑ O Git-Bash é distribuído como software livre e pode ser instalado a partir de um arquivo de instalação disponível em <https://git-scm.com/download/win>

Prática





Preparação

- ❑ Para a execução dos exercícios referentes ao conteúdo desta aula, será necessário que cada aluno crie uma conta no GitHub;
- ❑ Após a criação da conta no GitHub, baixe o instalador em <https://git-scm.com/download/win> e instale o Git-Bash em sua máquina;
- ❑ Agora, cada aluno deve fazer um fork do repositório: https://github.com/stefanomozart/monitoria_ciencia_computacao



Comandos Git

- ❑ Ao concluir esses passos, vamos utilizar os seguintes comandos git:
 - ❑ git init
 - ❑ git commit
 - ❑ git merge
 - ❑ git config
 - ❑ git clone
 - ❑ git push
 - ❑ git status
 - ❑ git branch
 - ❑ git pull
 - ❑ git add
 - ❑ git switch
- ❑ Agora, utilize os comandos acima para criar uma cópia local do repositório que acabou de clonar no GitHub, acrescente uma pasta com seu nome (ex. `mkdir StefanoMozart`) crie um arquivo `nota.txt` e escreva nele a nota que deseja receber nesta disciplina. Faça o **commit** dessas alterações e depois execute o comando **push origin**, para enviá-las para o GitHub;



Funcionalidades GitHub

- ❑ Ao concluir esses passos, utilize a funcionalidade de `pull request` no GitHub, para enviar suas alterações para o professor.

Dúvidas?

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.

COOL. HOW DO WE USE IT?

NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

