GitHub – Git-SCM (2005)

Sistema de Controle de Versão Distribuído

SCM – Source Control Management

- What changed?

- Why it was changed?

- Who made the changes?

- The changes can be reproduced?

Tópicos

Identificação

Documentação (Interna / Desenvolvimento)

Controle (Teste/Verificação)

Auditoria (Alinhamento com projeto)

Artefatos:

- Código Fonte

- Documentação do Software

- Manual de Usuário

**Controle de Versão:**

Exemplo: - Múltipla edição/sobrescrição de arquivos por diferentes agentes

Controle de versão possibilita o merge (junção)

Ferramentas de controle de versão:

- Subversion (SVN) (Mais antigo)

- Mercurial (Hg)

- CVS – Concurrent Versioning System (legado)

- Bazaar (pouco utilizado)

- Git mais rápido e eficiente

- Git tem execução distribuída (cada agente tem espelho do código)

- Possibilidade de execução sem conexão com servidor

# O que é o GitHub

Git – ferramenta de controle de versão

GitHub – um servidor de repositórios Git

Outros servidores:

GitHub – maioria dos projetos armazenados

Assinatura mensal para repositórios privados

GitLab – Tem repositórios privados sem necessidade de pagamento

Bitbucket – Suporta Git e Mercurial (repositório privado – limite 5 pessoas) (Não tem chave SSH)

SourceForge – Git/Mercurial/SVN

**Instalação:**

sudo apt-get install git

Escolher uma Interface Gráfica

Recomendação GitEye

<https://www.collab.net/products/giteye>

# Básico em Git

## Chave SSH

<https://help.github.com/en/github/authenticating-to-github/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent>

Ajuda

git help <command>

Documentação: <https://git-scm.com/>

# Comandos Git

## Git Add

Adicionar os aqruivos novos e modificados para o próximo commit:

git add <lista de arquivos>

Adicionar todos os arquivos na pasta para o próximo commit

git add .

Registrar o commit com todos os arquivos que usou git add

git commit [-m “Mensagem”]

Se o parâmetro Mensagem não for passado, abrirá um editor de texto para escrever a mensagem (Configurar o editor de texto)

git config –global core.editor gedit

git config --global core.editor "/Applications/Sublime\ Text.app/Contents/SharedSupport/bin/subl -n -w"

git config --global core.editor " /Applications/PyCharm CE.app/Contents/MacOS/pycharm -n -w"

<https://swcarpentry.github.io/git-novice/02-setup/>

## Git Workflow

Maior parte dos trabalhos consiste nestas tarefas

* Editar
* Commitar
* Soncronizar com o repositório remoto

Revisão de comandos

Status do repositórios – arquivos não adicionados/adicionaos ao commit

git status

Últimos commits realizados

git log

Adicionar arquivos para o commit

git add

Criar o commit

git commit

Enviar todos commits que ainda não foram enviados

git push

# Estado dos arquivos

Não monitorado

untracked

Modificado

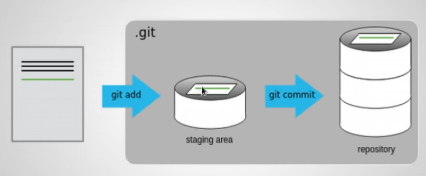
modified

Preparado (após git add)

staged

Consolidado (após o git commit)

commited



# Diff

Exibir diferenças entre commits e branchs

git diff

Diferenças em diretório específico

git diff [path]

Exibir alterações no último commit

git diff HEAD~1

# Aula 3

## Git Clone

Baixa o repositório remoto (cria o repositório com os dados remotos)

git clone

Outra forma de criar um repositório local

## Git Pull

Baixa as alterações do repositório remoto (Mantém o repositório sincronizado com os últimos commits de uma branch)

git pull

## Navegação no histórico

Permite verificar como um arquivo ou todo repositórios estava em um determinado commit

git checkout <commit> <file>

Retornar do modo detached head – para o estado atual (último commit)

git checkout máster

## Desfazendo alterações

Desfazer todas as alterações que não estejam no Stage (antes de git add) desde o último commit

git checkout -- <path\_or\_filename>

Para todos os arquivos:

git checkout -- .

Desfazer alterações desde o último commit incluindo o Stage (após git add)

Git checkout HEAD -- <path\_or\_filename>

## Git Revert

Reverte um commit – criando um commit “ao contrário” do commit realizado anteriormente

git revert <commit>

## Git reset

Reseta o repositório para um determinado commit

git reset <commit>

Resetar e remove todas as alterações (CUIDADO AO USAR! NÃO USAR SE JÁ ESTIVER PUBLICADO (enviado para GitHub – melhor usar revert))

git reset –hard <commit>

Útil para desfazer últimos commits antes do push

Reverter último commit:

git reset HEAD~1

Reverter últimos 2 commits (inclusive as alterações nos arquivos)

git reset HEAD~2 --hard

## Conflitos

Podem acontecer ao unirmos alterações.

Acontecem quando versões diferentes possuem as mesmas linhas nos mesmos arquivos editados.

O git identifica os conflitos e fica aguardando a solução deles.

Ao resolver os conflitos, deve ser feito um commit.

Tela de celular com texto preto sobre fundo branco

Descrição gerada automaticamente

Resolvendo conflitos com Merge

Ao realizar um commit de um mesmo arquivo editado o git rejeita e ao verificar o git status é possível verificar onde ocorreu o/os conflito/os.

Realizar a alteração do arquivo e depois o commit, que realizará o merge.

# Aula 4

## Branch

Lista de commits

Representa ramificações no repositório

Útil para trabalhos colaborativos

Facilitam o controle

Branch máster é a padrão

git branch

Asterisco indica qual branch se está atualmente

Criar uma nova branch

git branch <nova\_branch>

Excluir uma branch

git branch -d <branch>

Mudrar para a branch

git checkout <branch>

O repositório passa a ter os commits que a branch possui e novos commits serão adicionados a ela.

## Git Merge

Aplicar os commits de uma branch na branch em uso atualmente

git merge <branch>

Encontra um commit comum (base) entre as branchs e aplica todos os commits que a branch atual não possui.

Caso existam commits na branch atual que não estão na outra, será criado um commit de merge

Recomendado utilizar quando a branch divergiu muito (houver possibilidade de muitos conflitos) da branch master

Tela de celular com texto preto sobre fundo branco

Descrição gerada automaticamente

## Git Rebase

Semelhante ao merge, porém é diferente na ordem de aplicar os commits

git rebase <branch>

No Rebase, os commits na frente da base em <branch> são removidos temporariamente, os commits da branch atual são aplicados na sua branch atual e por fim os commits de <branch> são aplicados um a um, resultando em uma sequência linear.

Texto preto sobre fundo branco

Descrição gerada automaticamente

Pode acontecer conflitos que serão resolvidos para cada commit.

## Git Fetch

Baixa as atualizações do remote porém não as aplica no repositório

git fetch

Permite fazer o rebase de uma branch em vez de fazer o merge

Pull = Fetch + Merge

Fetch e Rebase é melhor para manter histórico do desenvolvimento

Sequência: realizar as alterações nos arquivos, realizar commit, realizar fetch, realizar rebase e solucionar possíveis conflitos, realizar push

## Tags

Útil para definir versões estáveis do projeto.

Semelhante a Branch, porém não recebe mais commits.

Guarda um estado de um repositório

git tag [nome da tag]

git push <remote> <tag>

# Aula 5 – Colaboração com Open Source

## Uso de Watching e Stars

## Uso de Forks:

<https://guides.github.com/introduction/flow/>

# Extras

# .gitignore

Arquivo utilizado para filtrar arquivos/caminhos que não devem ser monitorados/sincronizados pelo git.

Geralmente fica na raiz e os itens citados no arquivo não serão monitorados ou sincronizados:

Ex:

.project

\*\*/\*.css

.DS\_Store

.\_.DS\_Store

## Commit Amend

Altera o último commit realizado

git commit --amend

Possível alterar a mensagem, bem como adicionar arquivos. (para commits sem push)

## Stash

Guarda as alterações do Working Directory

git stash

Permite fazer rebase, merge, trocar de branch sem a necessidade de fazer um commit

Exibir uma lista dos stash’s

git stash list

Aplicar o último stash armazenado

git stash pop

Exemplo de uso – Quando um mesmo arquivo está sendo editado em branches locais diferentes, uma com commit e outra não. Possível aterar de uma branch para outra sem efetivamente realizar um commit.

## Cherrypick e Blame

Aplica as alterações de um commit na branch atual.

git cherry-pick <commit>

Cria um novo commit.

Útil para recuperar histórico.

Ex: Possibilita bugfix em versões antigas quando realizada uma alteração em uma versão mais atual que já divergiu bastante uma da outra. (pegandp um commit por vez)

Pegar o último commit

git cherry-pick novaBranch

Pegar o antepenúltimo commit

git cherry-pick novaBranch˜2

Mostrar as alterações feitas em um arquivo por linha

git blame

Mostra o autor e commit feito aquela linha.

Útil para verificar quando as alterações foram feitas, por que e por quem.

Ex:

git blame personagens

## Bisect

Permite fazer uma busca binária nos commits para encontrar uma alteração

git bisect

Útil para alterações que modificaram o comportamento e não podem ser identificadas por código facilmente. Ou quando a alteração é bastante antiga.

Ex:

Acessar a função bisect na branch atual:

git bisect start

Necessário informar qual commit está com problema (para commit atual):

git bisect bad

Necessário informar qual commit ainda não tem o problema:

git bisect good <commit>

Exibirá uma lista com os commits que alteraram entre os commits

Se no commit exibido o problema persiste: git bisect bad

Irá prosseguir para outro commit, caso o problema não exista: git bisect good

Para sair do modo bisect:

git bisect reset

Exercícios:

<https://github.com/git-game/git-game>

## GitHub Pages

Possibilita hospedagem de website estático

Site para usuários, organizações e repositórios.

Repositório: necessário branch ‘gh-pages’ com index.html

O GitHub tem um gerador de páginas. (Settings->GitHub Pages->Launch automatic page generator)

Possível checkout push para atualização dos arquivos no repositório do github pages.

Para o usuários criar um repositório que tenha o nome ‘wgadelha.github.io’ branch master deve ter o arquivo “index.html” – Bom para hospedar portfólio.

## GitHub Milestones

Grupo de Issues associados a um objetivo.

Possui uma data

Permite acompanhar a % de issues resolvidas

Possível atrelar novas issues a Milestones já criadas

<https://guides.github.com/features/issues/>

## WebHooks

Permite integrar eventos do repositório com outras aplicações

Possui alguns serviços já compatíveis

Eventos POST em JSON

Exemplo: Travis-CI (testes automatizados-JUnit) , Slack  
(Settings->WebHooks e services->

<https://github.com/integrations>

## GitKraken GUI

[www.gitkraken.com](http://www.gitkraken.com)