

# Git

Working Directory

git add

Staging Area

git commit

Git Directory

git init → inicia um repositório git no diretório atual.

git status → status atual do repositório.

git add → "trackear" os arquivos. Obs: git add adiciona todos os arquivos

git commit -m "mensagem do commit"

git commit -a -m "mensagem do commit" → não precisa fazer o git add

Nota: Arquivos que não terão versões controladas devem ser adicionado no arquivo .gitignore

arquivo.txt  
\*.gitignore  
↑  
lista de arquivos

arquivo .gitignore

arquivo oculto

esse arquivo tb pode ser ignorado

git diff → diferenças nos arquivos no working directory  
→ não dá p ver no staging Area

git diff --staged → ver as mudanças nos arquivos na staged área

git log → mostra todos os commits, (ordem dos mais recentes p/ os mais antigos)

git log -p → todos os commits com as mudanças

git log -p -2 → últimos dois commits.

Nota: gitk → interface com relatórios. Poderá ser instalado novos programas de visualização.

git log --pretty=oneline → mostra apenas as chaves

git commit --amend -m "Nova mensagem"

↳ altera o commit anterior. Poderia antes fazer "git add ." para colocar novas funcionalidades p/ acrescentar no commit anterior realizado por engano. Note que a chave do commit é mudada.

git reset HEAD arquivo.c → remove um arquivo da staging área.

git checkout -- arquivo.c → recupera o arquivo.c da forma do commit anterior.

git rm arquivo.c → remove um arquivo. Ou seja, reverte ao estado do último commit.

Obs: Usar tags ajuda a reverter o sistema a uma data e hora associada à tag.

git tag → lista as tags criadas. Nota: tags são ponteiros para o sistema.

git tag -a v1.0 -m "versão 1.0"

↳ nome da tag      ↳ mensagem  
↳ cria uma tag anotada, com informações do usuário, data e hora, etc.

git tag -a v0.0 chave -m "versão 0.0" → mudando ou add uma tag a commits anteriores.

(2)

git show v.0.0 → <sup>o nome da tag</sup> suber detalhes a respeito de uma tag.

git checkout v.0.0 → volta o sistema ao status da tag v.0.0

git checkout <sup>branch padrão</sup> master → volta para o último commit.

git tag -d v.0.0 → remove a tag v.0.0

git branch teste → criando um branch de nome "teste"

git checkout teste → mudando para o branch de nome "teste".

git checkout -b teste → cria o branch de nome "teste" e faz a troca de ambiente.

git merge teste → estando no branch master, estão trazendo e mesclando os arquivos do branch teste.

git branch -d teste → removendo o branch de nome teste

↳ alterações na mesma linha em branches distintos vão gerar conflitos. Na hora de fazer o merge, ocorrerá um problema (o merge não será feito!). Daí, deverá ir nos arquivos com conflitos e escolher o que irá ficar. No arquivo terá opções comentadas. Delete a que não quiser considerar!

git init --bare → cria o versionamento permitindo o compartilhamento em rede local.  
↳ no servidor. (fazer no servidor!)

git clone diretório do servidor (todo o caminho) novo nome  
↓  
nome do projeto clonado — nome na máquina local.  
do p/máquina local.

git remote → diz o nome do servidor. Note que isso é possível uma vez que estamos trabalhando em um clone (projeto clonado do servidor remoto). O padrão é "origin".

git push origin master → enviando os arquivos para o servidor origin que está no branch master.

↳ pega as atualizações do servidor que porventura foram feitas depois do clone e coloca no branch master. Se fizer o clone irá perder mudanças que fiz no clone localmente!

O pull faz um merge mas pode não ser legal. O melhor seria fazer atualizações para um branch diferente do master. Para isso, fazer:

git fetch origin teste

ssh-keygen → gerando uma chave de acesso p/ permitir que a máquina local venha interagir com o github. ↳ branch de nome "teste" na máquina local.

git reset HEAD~1 --hard → desfazendo o último commit e as alterações nos arquivos.

git reset HEAD~1 --soft → desfazendo o último commit. As alterações nos arquivos não serão perdidas.

↳ ~2 faz o mesmo penúltimo commit e assim por diante: ~3, ~4, ...

git push --delete origin tagname → remove  
o uma tag remotamente.

No github, fazer um fork <sup>ou</sup> é fazer um clone de  
projeto que <sup>queremos</sup> colaborar. Esse clone é  
feito no ambiente remoto do github. Depois, para  
se ter na máquina local, basta clonar com git clone.

Com os arquivos na máquina, basta fazer as alterações e  
e ao finalizar, enviamos ao fork do projeto em  
nossa conta. Depois, no github, devemos ir em  
"New pull request" e clicar para enviar ao autor  
original do projeto.