

Dicionário de Termos

PET Estatística UFPR

A

Add

Adiciona (envia) os arquivos para a stagin area, para que possa ser marcado no tempo por um commit.

Exemplo:

```
# Adicionar todos os arquivos a stagin area:  
git add *
```

B

Bisect

Realiza uma pesquisa binária (binary search) a procura de erros. Para que a pesquisa ocorra, é necessário um ponto no tempo em que o código esteja funcionando e outro que não esteja.

Exemplo:

```
# Pesquisa Binária:  
# Iniciando a pesquisa.  
git bisect start  
# Marcando o commit atual como não funcionando.  
git bisect bad  
# Marcando o commit com nome commit1 como funcionando:  
git bisect good commit1
```

Blame

É usado para pesquisar qual ‘commit’ modificou determinado arquivo, com o objetivo de encontrar quem e quando um erro foi inserido. Esse método também é chamado de anotação de arquivo.

Exemplo:

```
# Blame no arquivo teste.tex  
git blame teste.tex
```

Branch

É como uma cópia dos arquivos do ultimo commit para um ambiente de desenvolvimento paralelo, o que permite que as modificações em um ‘branch’ não afete os arquivos em outro. Os branches também são chamados de ramos de desenvolvimento. Veja com mais detalhes no capítulos dos workflows.

Exemplo:

```
# Cria um branch chamado novoBranch  
git branch novoBranch
```

C

Checkout

Serve para transitar entre branches e commits. Usando o checkout é possível voltar a commits anteriores.

Exemplo:

```
# Mudar do branch atual para o branch teste:  
git checkout teste
```

Clone

É usado quando deseja-se clonar um repositório que está disponível em um servidor remoto para o servidor local. Depois da clonagem, estará disponível todos os arquivos e todo o histórico de controle de versões sem a necessidade de uso da internet.

É importante salientar que quando se é usado o ‘clone’, o servidor remoto é adicionado automaticamente, podendo ser acessado através do nome ‘origin’.

Exemplo:

```
# Clonando o projeto desta apostila:  
git clone git@gitlab.c3sl.ufpr.br:pet-estatistica/apostila-git.git  
# Exibindo os servidores remotos:  
git remote
```

Commit

Marca os arquivos da stagin area como uma versão definitiva, para que posteriormente, caso algum erro ocorra, possamos voltar nos commits anteriores onde o código está em pleno funcionamento.

Exemplo:

```
git commit -m "Meu primeiro commit"
```

Config

É um comando usado para ajustar as configurações padrão do git. Há duas configurações básicas a serem feitas: a inclusão do e-mail e do nome do usuário Git. Todas as configurações definidas como globais ficam armazenadas em um arquivo chamado ‘gitconfig’, que fica localizado no diretório padrão do usuário.

Exemplo:

```
# Configurando o usuário Ezio Auditore:  
git config --global user.name "Ezio Auditore"  
# Configurando o e-mail:  
git config --global user.email ezio.auditore@exemple.com
```

F

Fetch

Atualiza o repositório local com as alterações do remoto, porém não realiza o merge dos arquivos, deixando isso para ser feito manualmente.

Exemplo:

```
# Buscando arquivos no servidor remoto origin:  
git fetch origin
```

H

HEAD

É um arquivo que contém um apontador para o ‘branch’ atual. Quando o ‘checkout’ é executado para a mudança do ‘branch’, esse arquivo é automaticamente modificado, apontando agora para o novo local, permitindo assim que, depois que o computador é desligado e reiniciado, o Git ainda esteja trabalhando com o mesmo ‘branch’.

Help

É a ajuda do git. Todo comando Git tem um manual de ajuda que pode ser exibido na tela com o comando ‘-help’.

M

Merge

Faz a fusão de dois ramos em um. Quando se trabalha em ramos diferentes (diferentes branches) e precisa-se posteriormente juntar o trabalho, o ‘merge’ (fundir) é usado, permitindo que o trabalho seja centralizado novamente. A fusão é feita de forma automática, mas conflitos podem ocorrer, como por exemplo, quando duas ou mais pessoas modificam a mesma parte do código. Estes conflitos devem ser resolvidos manualmente, deixando a cargo do gerente de projetos decidir que parte do código deve permanecer.

Exemplo:

```
# Faz merge do branch chamado novoBranch com o branch atual:  
git merge novoBranch
```

Mv

É usado para mover ou renomear arquivos.

Caso a mudança seja feita sem o ‘git mv’, o Git entende que o arquivo foi deletado e que um novo arquivo foi criado, deixando de fora, por exemplo, a ligação existente entre o arquivo e seus commits.

Exemplo:

```
# Renomeando o arquivo teste.tex para arquivo1.tex:  
git mv teste.tex arquivo1.tex
```

P

Pull

É semelhante ao comando 'fetch', porém, puxa os arquivos do servidor remoto fazendo merge. Caso haja algum conflito de merge, estes deverão ser resolvidos manualmente.

Exemplo:

```
# Puxando arquivos no servidor remoto origin:  
git pull origin
```

Push

É usado para "empurrar" os arquivos do repositório local para o servidor remoto.

Exemplo:

```
# Atualizado o branch master (remoto), com o branch atual (local):  
git push origin master
```

R

Rebase

É usado para modificar 'commits' antigos. Ele refaz a árvore de 'commits', sendo assim não é uma boa ideia fazer um 'push' da alteração, pois modificará a árvore do servidor afetando todos os desenvolvedores.

A ideia geral do 'rebase' é "mudar de base" os 'commits' de um ramo, passando-os para novos commits do ramo atual, formando uma árvore com fluxo de trabalho linear.

Exemplo:

```
# Fazer o rebase do branch teste para o atual:  
git rebase teste
```

Rm

De forma mais prática serve para remover um arquivo, que deixa de ser gerenciado pelo Git e é excluído do disco rígido. Também é possível que o arquivo deixe de ser gerenciado e permaneça no disco.

Uma das vantagens é que, quando o arquivo é removido pelo ‘git rm’, já aparece como preparado (staged), precisando somente que a exclusão seja commitada.

Exemplo:

```
# Remover arquivo teste.tex do gerenciamento e do disco:  
git rm teste.tex  
  
# Remover arquivo teste.tex apenas do gerenciamento:  
git rm --cached teste.tex
```

Remote

É o servidor remoto onde os arquivos Git estão hospedados. O remote padrão geralmente é criado com o nome de ‘origin’, mas é possível adicioná-lo utilizando outros nomes e até mesmo adicionar outros servidores remotos juntamente ao ‘origin’.

Exemplo:

```
# Adicionando um servidor remoto com nome origin:  
git remote add origin "git@gitlab.c3sl.ufpr.br:pet-estatistica/apostila-git.git"
```

Repositório

É uma pasta gerenciada pelo git. A partir da criação desta, usufruímos do sistema de versionamento, sendo possível transitar entre as diferentes versões dos arquivos a medida que necessário.

Exemplo:

```
# Iniciar repositório na pasta atual:  
git init
```

Reset

Enquanto o ‘git checkout’ somente transita entre os ‘commits’, o ‘reset’ pode também alterar o histórico, fazendo ‘commits’ serem apagados de maneira irreversível (–hard) ou serem apenas resetados ao estado de não commitado (–soft).

Exemplo:

```
# Apagando o último commit (voltando ao anterior):  
git reset --hard HEAD~1
```

S

SSH Key

É uma chave de autenticação baseada em criptografia de chave pública (chave assimétrica). Essa criptografia torna o processo de transferência de arquivos entre o cliente e o servidor mais segura.

Para que o git local consiga se conectar a um servidor git remoto, é necessário que esta chave seja gerada e que as devidas configurações sejam feitas tanto localmente quanto no servidor remoto, caso contrário, é exibido um erro e a conexão é interrompida.

Exemplo:

```
# Gerando chave SSH:  
ssh-keygen
```

Stagin Area

É um espaço temporário na pasta gerenciada pelo Git. É o local em que ficam os arquivos antes de serem marcados como uma versão definitiva. Em tradução livre, stagin area é área de estágio, podemos assim imaginar que o arquivo está estagiando antes de ser promovido a um arquivo definitivo.

Stash

É um comando usado para não ser necessário fazer um ‘commit’ para mudar de ‘branch’. Ao executá-lo, os arquivos modificados ficam salvos em uma pilha de modificações inacabadas, sendo possível transitar entre ‘branches’ e voltar ao trabalho inacabado quando necessário.

Exemplo:

```
# Fazendo o stash:  
git stash  
# Listando os stash criados:  
git stash list
```

Status

Exibe a diferença entre o estado atual dos arquivos e o estado do último ‘commit’ do mesmo ‘branch’. São três estados possíveis: consolidado (committed), modificado (modified) e preparado (staged).

Exemplo:

```
# Pedindo o status:  
git status
```

T

Tag

É usada para marcar pontos específicos do desenvolvimento. Geralmente tags são usadas para marcar versões definitivas, como a v1.0, v2.0 e assim por diante.

É dividida em dois tipos: leve e anotada. A ‘tag’ leve simplesmente aponta para um ‘commit’ específico. Já a ‘tag’ anotada é guardada como objetos inteiros, possuindo algumas informações, como o nome da pessoa que criou, a data, uma mensagem semelhante a de ‘commit’, entre outras.

Exemplo:

```
# Criando tag leve:  
git tag -l "v1.0.0"  
  
# Criando tag anotada:  
git tag -a v1.0 -m "Minha primeira tag anotada."
```