Curso de Introdução ao Git

USPGameDev

27 de agosto de 2012

Introdução

O que é Controle de Versão?

Softwares de controle de versão é um sistema que grava todas as modificações em um conjunto de arquivos, criando um histórico deles, criando um ambiente onde o usuário possa recriar versões antigas dos arquivos.
Eles são separados em 3 conjuntos:

- Controle de versão local. (Ex: rcs)
- Controle de versão centralizado. (Ex: SVN, CVS)
- Controle de versão distribuído. (Ex: Git, Mercurial)

Fundamentos

Snapshots, não diferenças

O Git, diferente de outros controles de versões que salvam os arquivos e as modificações entre os commits, o Git salva, a cada commit, o estado atual de cada arquivo.

Trabalho Local

Quase todo trabalho do Git é local. Isso significa que ele não precisa de comunicação constante com algum servidor remoto, como é normal em outros CVs. Essa liberdade garante que os usuários não fiquem dependentes de internet ou intranet para trabalhar.

Integridade

Todo arquivo que está sob os cuidados do Git recebe um checksum (soma de verificação). Olhando para esse checksum, o Git sabe se algum arquivo ou diretório foi modificado, tornando impossível modificações obscuras.

Adição de dados (geralmente...)

Quase todas as ações do Git tem como finalidade adicionar dados na database do projeto. Depois que algo foi incorporado em um repositório, é quase impossível tirá-lo de lá.

3 Estados

Todo e qualquer arquivos que está no Git tem 3 possíveis estados:

- Committed - Significa que as modanças feitas no arquivo estão salvas na database do Git.
- Modificado - Significa que o arquivo foi modificado.
- Staged - Significa que as mudanças atuais no arquivos seram inclusas no próximo commit.

3 Estados

Esses 3 estados fazem com que um repositório local do Git seja separado em 3 seções:

- Git directory - Onde todos os arquivos vitais e a database do Git estão.
- Working directory - É uma instância de uma versão do seu projeto.
- Staging area - Um arquivo que contêm todas as modificações que entrarão no próximo commit.

3 Estados

O processo de trabalho com o Git pode ser resumido a:

1. Você modifica os arquivos no seu working directory.
2. Você adiciona as modificações na staging area.
3. Você commita, aicionando permanentemente as modificações no seu Git directory.

Instalação

Para instalar o Git, vamos seguir o tutorial do GitHub =D

Configuração

O Git permite uma variedade de configurações. Podemos modificá-las usando o comando:

```
$ git config
```

Configuração

O Git permite uma variedade de configurações. Podemos modificá-las usando o comando:

```
$ git config
```

Duas configurações que temos que modificar depois de instalar o Git são:

```
$ git config --global user.name "José da Silva"
$ git config --global user.email josé@silva.com.br
```

Temos que mudar essas configurações pois o Git usam elas para gerar as mensagens de commit.

Iniciando um Repositório

Podemos pegar um projeto que usa o Git de duas maneiras distintas:

- Portando um projeto existente para o Git, ou
- Clonando um projeto que já usa o Git.

Iniciando um Repositório

Para iniciar o Git num projeto já existente basta ir na pasta raiz do projeto e usar o comando:

```
$ git init
```

Iniciando um Repositório

Para iniciar o Git num projeto já existente basta ir na pasta raiz do projeto e usar o comando:

```
$ git init
```

Para clonar um projeto que já usa o Git usamos o comando:

```
$ git clone <url-do-repositorio> <nome-da-pasta>
```

Esse comando cria uma pasta com o nome do projeto no local em que foi chamado. Podemos passar um nome para a pasta que vai seer criada.

Tracked e Untracked

Agora que temos um repositório funcionando, vamos entender como o Git grava as mudanças.

Tracked e Untracked

Agora que temos um repositório funcionando, vamos entender como o Git grava as mudanças.
Lembram que todo arquivo no Git tinha 3 estados?

Tracked e Untracked

Agora que temos um repositório funcionando, vamos entender como o Git grava as mudanças.
Lembram que todo arquivo no Git tinha 3 estados?
Podemos adicionar mais dois estados gerais, **Tracked** e **Untracked**.

Tracked e Untracked

Agora que temos um repositório funcionando, vamos entender como o Git grava as mudanças.
Lembram que todo arquivo no Git tinha 3 estados?
Podemos adicionar mais dois estados gerais, **Tracked** e **Untracked**.
Os arquivos **Tracked** estão na database do Git, e eles podem estar em qualquer dos 3 estados citados anteriormente.

Tracked e Untracked

Agora que temos um repositório funcionando, vamos entender como o Git grava as mudanças.
Lembram que todo arquivo no Git tinha 3 estados?
Podemos adicionar mais dois estados gerais, **Tracked** e **Untracked**.
Os arquivos **Tracked** estão na database do Git, e eles podem estar em qualquer dos 3 estados citados anteriormente.
Os **Untracked** são os arquivos que não estão na database do Git, ou seja, o Git não grava as mudanças feitas neles.

Tracked e Untracked

Git status

Mas como sabemos em qual estado estão os arquivos?

Git status

Mas como sabemos em qual estado estão os arquivos?
Podemos usar o comando status do Git, que tem essa saída:

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   new file:   README
#
```

Mais git status

Agora vamos analisar a saída do comando status...

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..."
        to unstage)
#
#   new file:   README
#
# Changed but not updated:
#   (use "git add <file>..."
        to update what will be committed)
#
#   modified:   benchmarks.rb
```

Git add

O comando git add tem duas principais funcionalidades:

- Adicionar um novo arquivo na database do Git. O arquivo passa de Untracked para Tracked.
- Adicionar uma nova versão de um Arquivo Tracked na database. O arquivo passa de modified para staged.

A sintaxe é simples:

```
$ git add <nome-do-arquivo>
```

Git rm

Assim como podemos adicionar arquivos na database, podemos removê-los.

Git rm

Assim como podemos adicionar arquivos na database, podemos removê-los.
O comando rm do Git remove os arquivos tanto do workdirectory como da database.

Git rm

Assim como podemos adicionar arquivos na database, podemos removê-los.
O comando rm do Git remove os arquivos tanto do workdirectory como da database.
Temos que tomar cuidado ao deletar um arquivo sem o git rm.

.gitignore

Nem sempre é bom que o Git veja todos os arquivos dentro de um projeto.

.gitignore

Nem sempre é bom que o Git veja todos os arquivos dentro de um projeto.
Imagine se toda vez que o comando status lista a saída, temos vários arquivos de compilação lá?

.gitignore

Nem sempre é bom que o Git veja todos os arquivos dentro de um projeto.
Imagine se toda vez que o comando status lista a saída, temos vários arquivos de compilação lá?
Podemos colocar expressões regulares num arquivo chamando **.gitignore**.

.gitignore

Nem sempre é bom que o Git veja todos os arquivos dentro de um projeto.
Imagine se toda vez que o comando status lista a saída, temos vários arquivos de compilação lá?
Podemos colocar expressões regulares num arquivo chamando **.gitignore**.
Todo arquivo que case com qualquer expressão do **.gitignore** será ignorado pelo Git.

.gitignore

Nem sempre é bom que o Git veja todos os arquivos dentro de um projeto.
Imagine se toda vez que o comando status lista a saída, temos vários arquivos de compilação lá?
Podemos colocar expressões regulares num arquivo chamando **.gitignore**.
Todo arquivo que case com qualquer expressão do **.gitignore** será ignorado pelo Git.
O **.gitignore** se aplica na pasta que ele se encontra, e em todas abaixo dele.

Commits

Certo. Agora que sabemos como adicionar as mudanças para a area staged.
Mas como passamos essas modificações para a database?

Commits

Certo. Agora que sabemos como adicionar as mudanças para a area staged.
Mas como passamos essas modificações para a database?
Usamos os commits para levar essas modificações até a database.
Para commitar suas mudanças, usamos o comando:

```
$ git commit
```

Esse comando abre uma tela de edição onde escrevemos a mensagem de commit.

Boas práticas de commit

Ao commitar, é bom seguir algumas práticas:

- Faça commits pequenos. Isso cria um histórico mais rico.
- Crie mensagens que expressem seu commit. Isso ajuda a identificar as mudanças.

Commits

O commit leva consigo algumas informações:

- As modificações que foram introduzidas nesse commit.
- O nome e o e-mail de que fez o commit.
- Um HASH que identifica o commit de maneira unica.

Movendo arquivos

Para mover arquivos dentro de um repositorio Git, usamos o comando:

```
$ git mv
```

Movendo arquivos

Para mover arquivos dentro de um repositorio Git, usamos o comando:

```
$ git mv
```

Se movemos um arquivo usando só o `mv`, o Git percebe que o arquivo foi movido mas não adiciona a movimentação automaticamente na staging area.

Git log

Para acessar o historico de commits usamos o comando:

```
$ git log
```

Git log

Para acessar o historico de commits usamos o comando:

```
$ git log
```

O comando devolve uma lista que mostra todos os commits desde o começo do repositorio.
Cada elemento da lista contêm as informações dos commits (as mesmas informações que os commits levam consigo).

–amend

O comando commit aceita a opção –amend, que possibilita a edição do commit que acabou de ser feito.
Podemos colocar mais arquivos no commit ou mudar a mensagme do mesmo.
A sintaxe é:

```
$ git commit --amend
```

git reset HEAD

Para retirar arquivos da area staged, usamos o comando:

```
$ git reset HEAD <arquivo>
```

git checkout –

Para reverter as modificações feitas em um arquivo que ainda não foi adicionado a database do Git, usamos o comando:

```
$ git checkout -- <arquivo>
```

Remotos

Repositórios remotos são versões do seu repositório local que ficam ou na internet ou em alguma máquina na rede.

Remotos

Repositórios remotos são versões do seu repositório local que ficam ou na internet ou em alguma máquina na rede.
O seu repositório local reconhece os remotos que ele conhece por aliases. Podemos ver esses aliases usando o comando:

```
$ git remote -v
```

Remotos

Repositórios remotos são versões do seu repositório local que ficam ou na internet ou em alguma máquina na rede.
O seu repositório local reconhece os remotos que ele conhece por aliases. Podemos ver esses aliases usando o comando:

```
$ git remote -v
```

Quando clonamos um repositório, o Git cria um alias chamado origin que faz referência a url "clonada".

Adicionar, Remover e Renomear

Para adicionar um novo remote:

```
$ git remote add <alias> <url>
```

Adicionar, Remover e Renomear

Para adicionar um novo remote:

```
$ git remote add <alias> <url>
```

Para remover um remote:

```
$ git remote rm <alias>
```

Adicionar, Remover e Renomear

Para adicionar um novo remote:

```
$ git remote add <alias> <url>
```

Para remover um remote:

```
$ git remote rm <alias>
```

Para renomear um remote:

```
$ git remote rename <alias-velho> <alias-novo>
```

Pushando

Para pushar para um remote, usamos o comando:

```
$ git push <remote-alias> <nome-da-branch>
```

O que são branchs

Branchs são separações no fluxo de commits de um repositório.

O que são branchs

Branchs são separações no fluxo de commits de um repositório. Usamos elas para trabalhar em várias partes do projeto sem influenciar no fluxo principal do mesmo.

Criando Branchs

Quando criamos uma branch, ela usa o commit atual da branch atual como base por default.
Usamos 2 comandos para criar uma branch:

```
$ git checkout -b <nome-da-branch> ou
$ git branch <nome-da-branch>
```

Listando, mudando e deletando de branchs

Podemos listar as branchs atuais usando o comando:

```
$ git branch
```

A branch que estiver marcada com um * é a branch atual.

Listando, mudando e deletando de branchs

Podemos listar as branchs atuais usando o comando:

```
$ git branch
```

A branch que estiver marcada com um * é a branch atual.
Para mudar de branchs usamos o comando:

```
$ git checkout <nome-da-branch>
```

Listando, mudando e deletando de branchs

Podemos listar as branchs atuais usando o comando:

```
$ git branch
```

A branch que estiver marcada com um * é a branch atual.
Para mudar de branchs usamos o comando:

```
$ git checkout <nome-da-branch>
```

Para deletar uma branch, o comando é:

```
$ git branch -d <nome-da-branch>
```

Mandando branchs para o remoto

Para mandar uma branch para um repositório remoto, o comando push é usado:

```
$ git push -u <remote> <branch>
```

Merge

Para passar as mudanças de uma branch para outra, usamos o comando:

```
$ git merge <nome-da-branch>
```

É na branch em que esse comando é rodado que a junção das modificações ficara.