Curso de git

Aula 1

Pet Computação

22 de outubro de 2018

Conteúdo

Aula 1

- Controle de versão
- Git conceitual
- Git prático com:
  - git add, commit
  - git checkout, branch, log

Aula 2

- Ramificações remotas
  - git pull, fetch e push
  - rebase
- Markdown para repositórios remotos
- Funcionalidades de alguns servidores remotos como:
  - Issues
  - Wiki
  - Interpretação de commit
- Boas práticas de contribuição
- Projeto

Referências

O material do curso foi construído pelo material do site git-scm. Foi utilizado também a segunda versão do livro Pro Git que é *open source*.

Controle de versão

O que é?

Controle de versão é um sistema de monitoramento de modificações em arquivos de determinado projeto.

Qual a utilidade?

Versionamento é importante para saber quem fez ou está fazendo no projeto, ter a possibilidade de restaurar determinada versão do projeto e também organizar o projeto.

Tipos de controle de versão

- Local;
- centralizado;
- Distribuído;

Controle de versão local

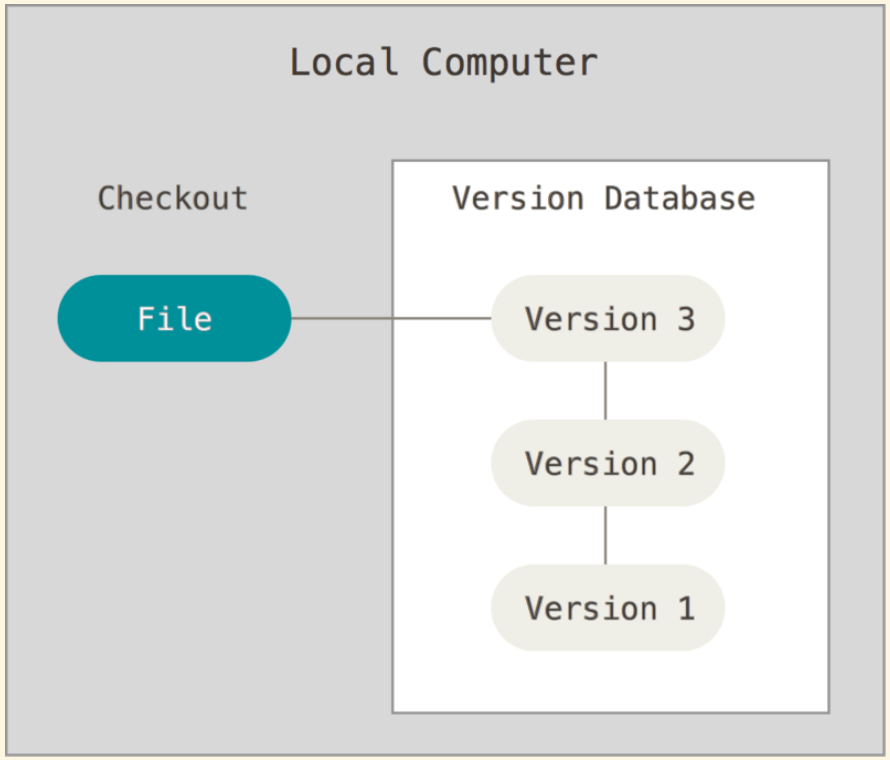

Figura: Diagrama do sistema de versionamento local.

Controle de versão centralizado

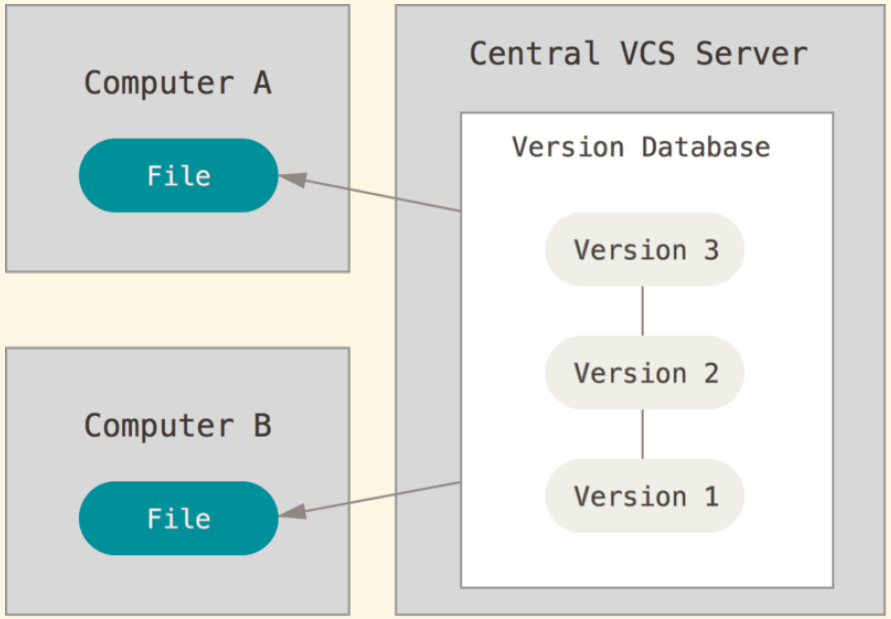

Figura: Diagrama do sistema de versionamento centralizado.

Controle de versão Distribuído

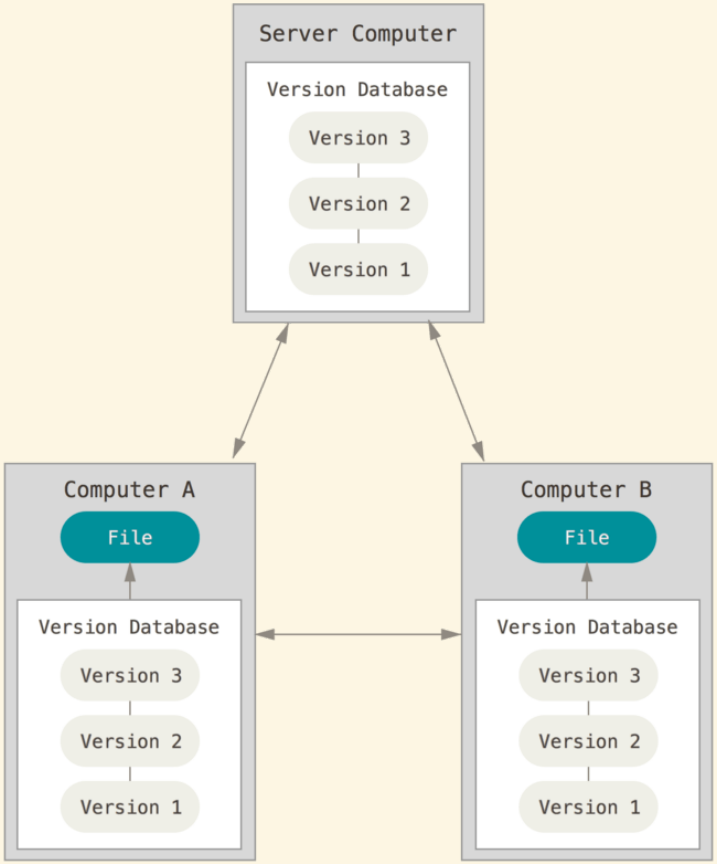

Figura: Diagrama do sistema de versionamento distribuído,

Git

Git é um programa de versionamento distribuído com os seguintes ideais:

- Velocidade
- Simples
- Integridade

História do git

O git nasceu após divergência de uso do programa de versionamento do kernel do linux, BitKeeper. Após inúmeros problemas de desempenho e o Bitkeeper se tornar pago o criador do linux, Linus Torvalds, resolveu criar um sistema de versionamento eficiente e prático para versionar o Linux. O projeto do Linus Torvalds deu tão certo que hoje o Git é uns dos mais populares software de versionamento distribuído.

Git

Snapshots

A ideia básica do git é tirar uma "foto"do projeto a cada nova alteração do projeto, assim mantendo um histórico do desenvolvimento.

Branch

O git é um sistema distribuído no qual várias pessoas podem trabalhar em diferentes itens e no final juntar todas as partes de forma eficiente.

Repositórios remotos

Possibilidade de distribuir facilmente um projeto entre a equipe e terceiros.

Três estados básicos

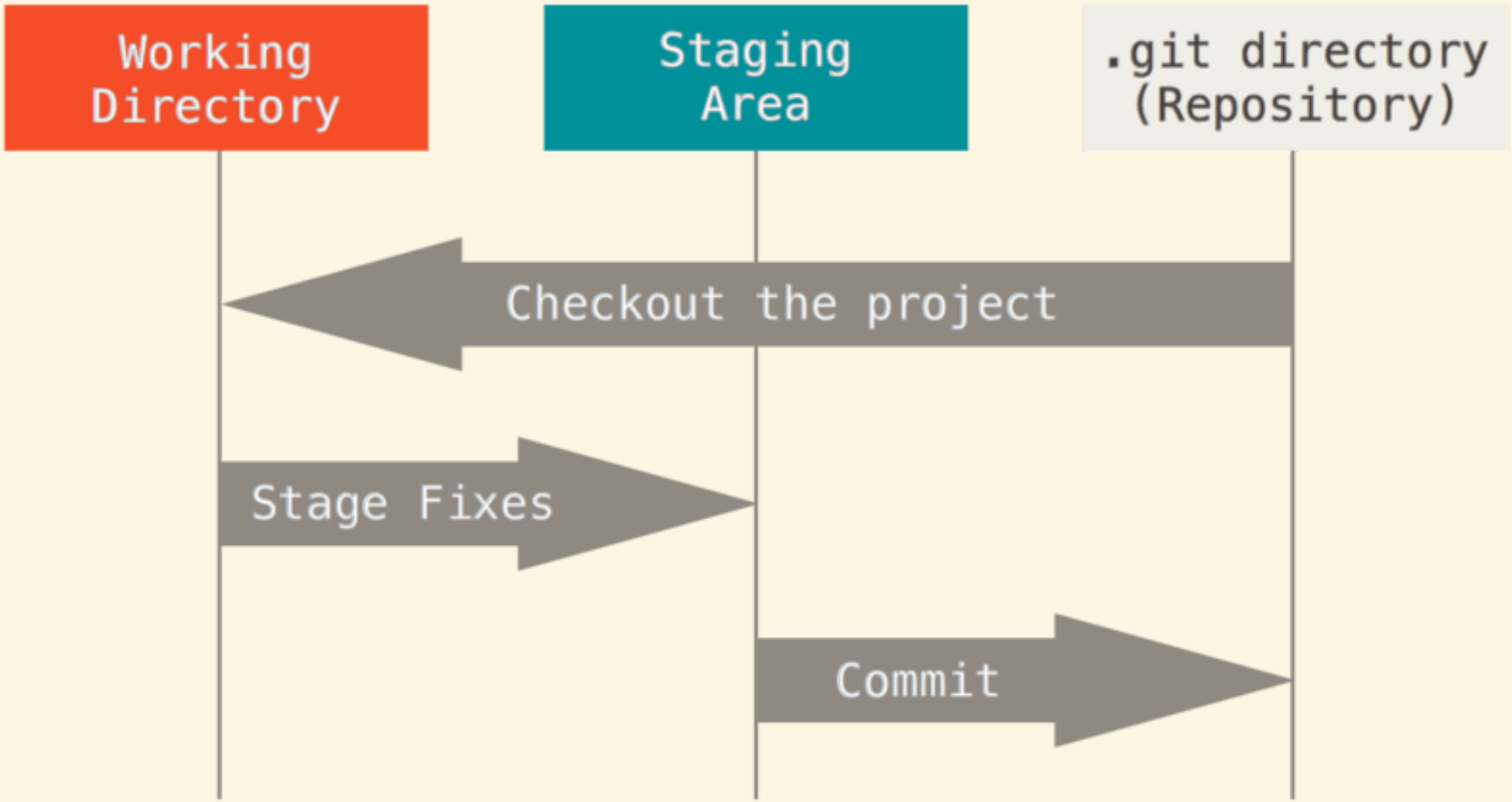

Figura: Caption

Instalação do git Linux

Distribuição baseada no Debian

Instalação para sistema operacionais como Debian, Ubuntu, Linux Mint e etc.

```
sudo apt-install git
```

Distribuição baseada no Fedora

```
yum install git-core
```

Distribuição baseada no Arch Linux

```
sudo pacman -S git
```

Instalação do git Mac Os

Instalador gráfico

Link

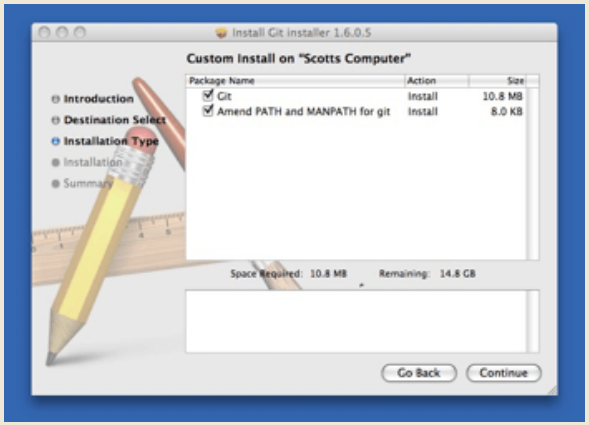

Figura: Instalação do git pela interface gráfica do Mac Os

Instalação pelo Brew

```
brew install git
```

Instalação do git Windows

A instalação do git no windows é puramente grafica e sucessivos 'nexts', as introduções estão neste link: Instalação Windows.

Configurações básicas

Após a instalação do git é nessário configurar algumas coisas como nome e email que aparecerão ao fazer alterações em arquivos.

Configurando identidade

Quando o git vai guardar alterações feita no repositório ele guarda nome e email de quem alterou.

```
git config --global user.name "Odair␣M"
git config --global user.email "omdj17@ufpr.br"
```

Configurando editor de text

Algumas comandos do git necessitam de inserção de mensagems, e por padrão o git abre um editor de texto para o usuário inserir a mensagem. É possível configurar qual editor de texto o git vai abrir.

```
git config --global core.editor vim
```

Iniciando um repositório

Existem dois modos de inicializar um repositório git. O primeiro é inicializar um repositório vazio com `git init` e o segundo modo é clonando um repositório existente com `git clone`

git init

Dentro de uma pasta que pode ou não conter um projeto, inicialize o git com:

```
mkdir curso_git && cd curso_git;
git init;
```

git clone

Para obter um repositório git existente, existe o mecanismo de clone no qual o usuário faz o download do projeto com todo o monitoramento do git existente.

```
git clone https://gitlab.c3sl.ufpr.br/pet/Farol
```

Ciclos de um arquivo

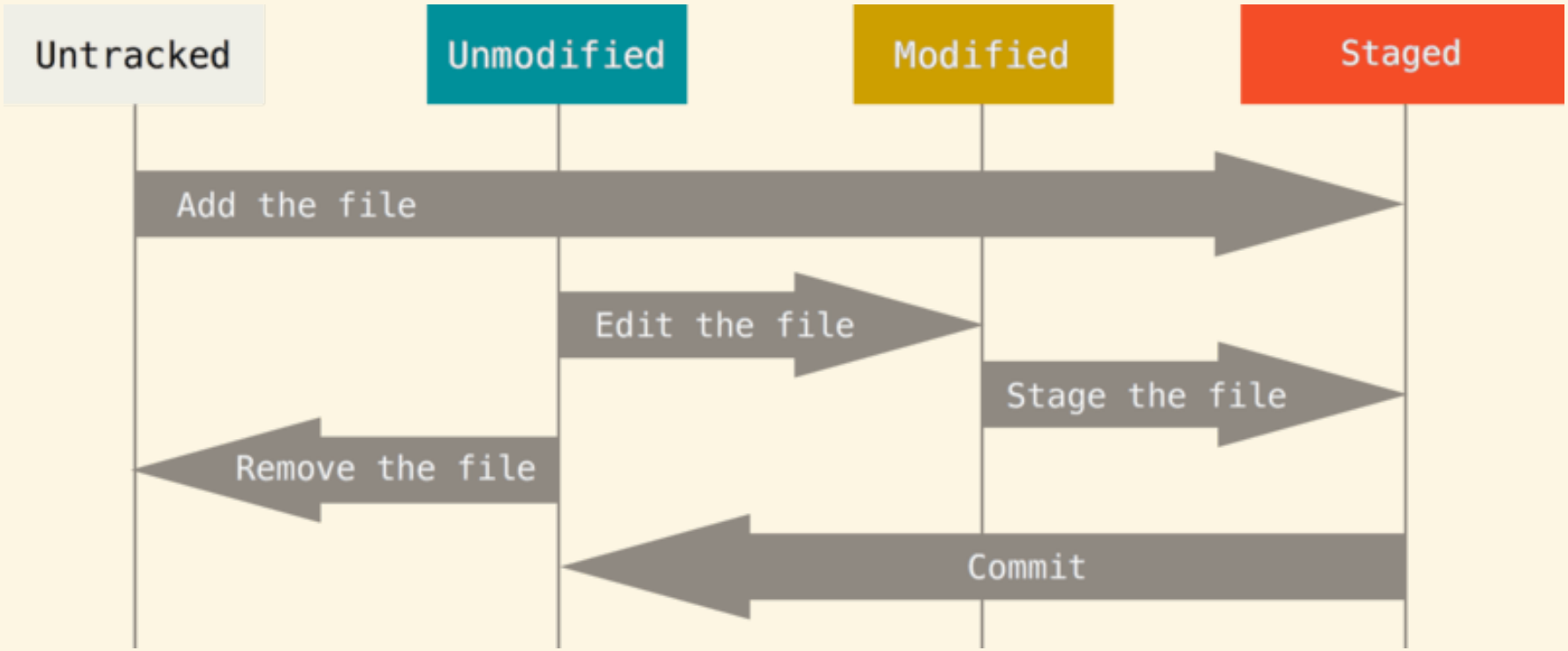

Figura: Ciclos de estados de um arquivo

Status de um arquivo

Ao longo do desenvolvimento do projeto é necessário saber qual o estado atual de um determinado arquivo. Por exemplo se o arquivo está sendo monitorado, se o arquivo foi modificado ou se existe arquivos na área de transição.
O comando `git status` mostra se existe arquivos não monitorados, modificados ou em transição.

Exemplo

```
touch {a..d}; git add b c d;
git commit -m "adiciona␣b␣c␣d";
echo "pedro" > b;
echo "odair" >> c; git add b;
rm d;
git status;
```

Opções do git status

-s –shot

`git status -s` ou `git status --shot` mostra git status de forma resumida.

-s –shot

`git status -s` ou `git status --shot` mostra git status de forma resumida.

-b –branch

`git status -b` ou `git status --branch` mostra status informaçoes da branch atual.

–show-stash

`git status --show-stash` olhar a definição oficial.

git add

O comando `git add` possui duas principais funções, a de começar a monitorar arquivos e a de adicionar arquivos na aŕea de transição.

Monitorando novos arquivos

```
touch nomes; git status;
git add nomes;
git status
git commit -m "adicionado␣arquivo␣nomes";
git status
```

Adicionando arquivos na área de transição

```
echo "Maria" >> nomes; git status;
git add nomes;
git commit -m "insere␣'maria'␣em␣nomes"
git status
```

Opções git add

-A –all

Adiciona todos os arquivos não monitorado ou modificado à área de transição, é equivalente a `git add .`. Não é recomendado adicionar todos os arquivo de uma única vez, pois pode adicionar arquivos binários ou temporários ao repositório.

-f –force

Força a adição de um arquivo não monitorado a área de transição.

git ignore

Ao longo do desenvolvimento de um projeto é normal surgi alguns arquivos temporários que não devem

Exemplo

```
touch .gitignore && ls;
echo "*.tmp" >> .gitignore;
git add .gitignore;
git commit -m "ignora␣arquivos␣.tmp";
touch ignore_me.tmp && git add ignore_me.tmp;
```

git commit

Consolida as alterações nos arquivos que estão na área de transição.

commitando com editor

```
echo "joao" >> nomes; git add nomes;
git commit;
```

commitando pela linha de comando

```
echo "eduardo" >> nomes; git add nomes;
git commit -m "adicionando␣eduardo␣a␣nomes";
```

Opções do git commit

-a –all

`git commit -a` ou `git commit --all` adiciona todos os arquivos modificados para a area de transição e realiza commit. Uma possível simplificação da sequencia de comandos:

```
git add -A;
git commit -m "uma␣mensagem";
```

por:

```
git commit -am "uma␣menssagem";
```

-C commit — — *reuse* — *message* = *commit*

Reutilizar uma mensagem de commit anterior.

```
git commit -C <33m227351eESC>;
```

git rm e git mv

git rm

`git rm <arquivo>` remove um arquivo do repositório e do projeto

```
git rm a;
git status;
```

git mv

Mover arquivos com o comando bash `mv <arquivo> <destino>` pode causar algumas inconsistência no repositório, o comando `git mv <arquivo> <destino>` move o arquivo e garante que não haja inconsistências.

```
git mv b a;
git commit -a -m "movi␣'b'␣para␣o␣'a'";
git status;
```

Histórico de alterações

A grande motivação de se utilizar softwares de controle de versão é manter um histórico das modificações feitas no projeto. O git oferece algumas motivações de poder olhar histórico de alterações e restaurar determinadas modificações.

git log

O log do git consiste nos logs dos commits.

git log simplificado

```
git log
```

git log interessante

Simplificação do log.

```
git log --oneline
```

Gráficos de commits e merges com git log

```
git log --graph
```

Desfazendo as coisas Parte 1

A ideia de controle de versão é ter a possibilidade de saber quais alterações foram feitas e se necessário retornar a uma versão anterior do projeto. O git fornece algumas ferramentas para desfazer de maneira eficiente alterações passadas. Nesta primeira parte será mostrado como desfazer alterações em arquivos modificados, arquivos que estão na área de transição e atualizar o último commit.

Desfazendo alterações em arquivos modificado

De acordo com imagem Ciclo de arquivos, arquivo modificado é um arquivo que está sendo monitorado, foi modificado mas não foi posto na área de transição. Para desfazer todas as alterações feitas se utiliza o comando `git checkout -- <arquivo>`.

Exemplo

```
echo "Pedro" >> nomes; git status;
git checkout -- nomes;
```

Desfazendo alterações de arquivos na área de transição

Em alguns momentos pode acontecer de um arquivo ir para área de transição de forma prematura, para retira um arquivo da área de transição utilize o comando `git reset HEAD <arquivo>`.

Exemplo

```
echo "0" >> nomes; git add nomes ; git status;
git reset HEAD nomes;
```

E caso seja necessário desfazer alterações no arquivo faça:

```
git checkout -- nomes;
```

Atualizando o último commit

Ao longo de desenvolvimento de um projeto versionado com o git é comum realizar um commit e esquecer de adicionar um arquivo ou errar a mensagem de commit. Para estes problemas existe o comando `git`commit –amend que dependendo do contexto adiciona arquivos a um commit ou altera a mensagem do commit.

Modificando a mensagem do último commit

```
echo "omdj17" >> nomes; git add nomes;
git commit -m "adiciona␣"; git log --oneline;
git commit --amend -m "adiciona␣login␣omdj17";
```

Atualizando o último commit

Adicionando um arquivo no último commit

```
touch login; echo "omdj17" >> login;
git add login;
git commit --amend;
```

Ramificações

Ramificações visões diferente do projeto que permitem várias pessoas contriburem para um mesmo projeto, sem um interferir no trabalho no progresso de outro membro da equipe.

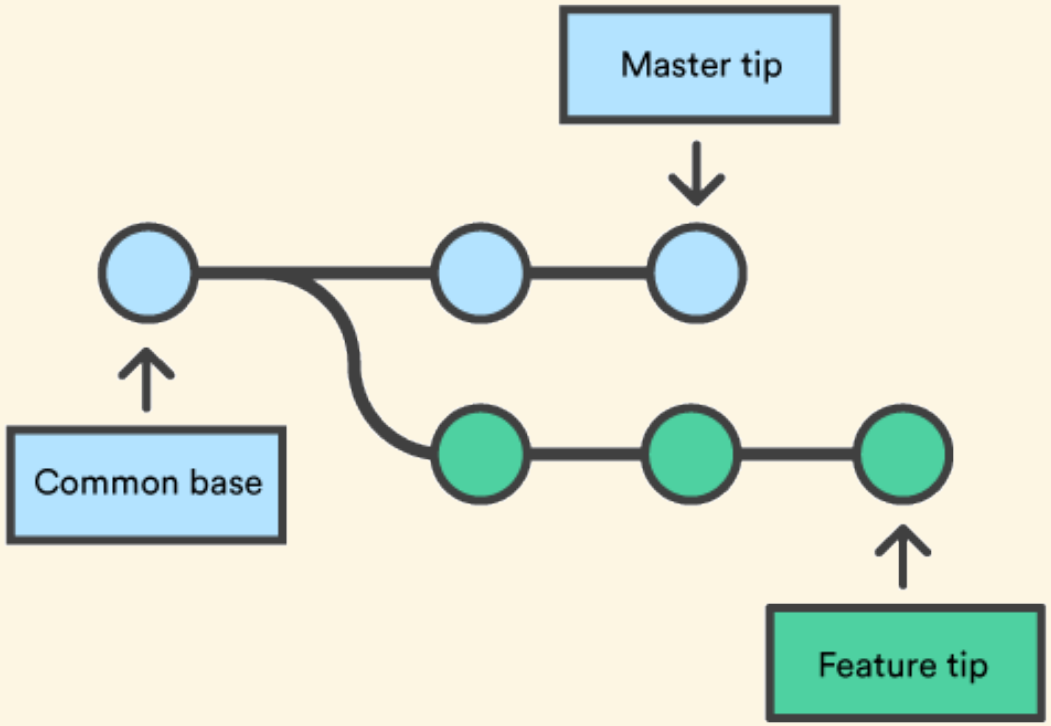

Figura: Ramificação

Unindo duas ramificações

Após trabalhar em sua ramificação é necessário mesclar com a ramificação principal do projeto.

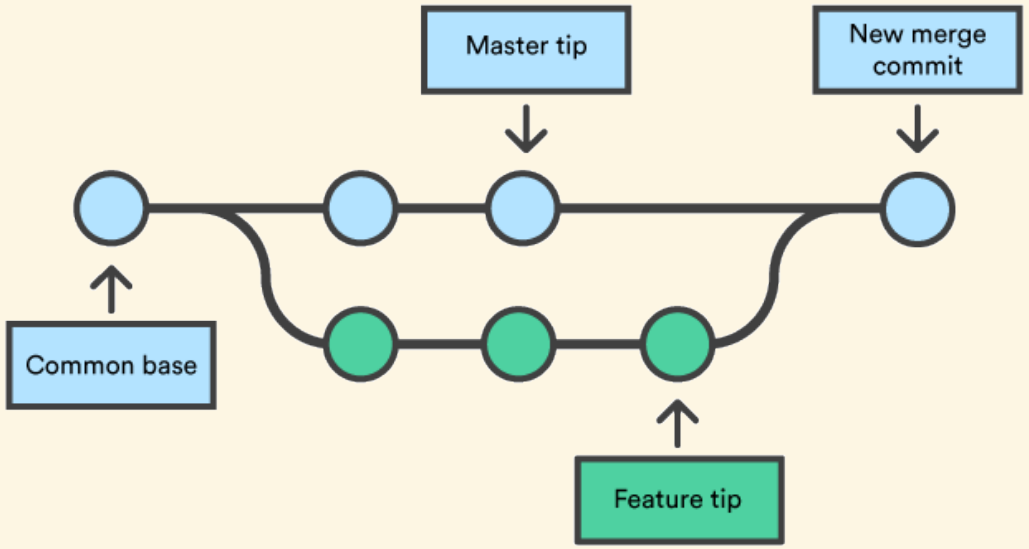

Figura: Unindo duas ramificações

git branch

Criando branch

```
git branch <nome branch>
```

ou

```
git checkout -b <nome_branch>
```

git merge

Para unir duas branch, mude para branch destinatária e faça:

```
git merge <nome_branch>
```

Resolvendo merge conflict

Eventualmente ao se dá merge de duas branchs vai acontecer de o git não conseguir mesclar as duas branchs e você terá que fazer isso manualmente.

Merge manual

Para cada arquivo que o git não conseguir mesclar ele coloca as duas versões do arquivo separado por <<<< indicando alteração da branch atual, ==== separando as duas versões e >>>> indicando o término da versão da branch que foi tentado o merge.

Gerenciando Ramificações

Listando branchs

Para listar todas as branchs locais e remotas utilize o comando `git branch -a` ou `git branch --all`

Deletando branchs

Existem duas formas de deletar uma branch com o comando `git branch -d <nome da branch>` que deleta a branch se e somente se o conteúdo da branch já foi unido com outra branch, caso contrário o git não permite remover. O outro comando para remover uma ramificação é `git branch -D <nome da branch>` que remove a ramificação independente do conteúdo ser unido a outra branch não.

Boas práticas com ramificações

Existem algumas boas práticas ao utilizar branchs como:

Duração da branch

O ideal é ter ramificações de curta duração em relação a master. Ter uma branch que está a muito tempo sem atualização da branch principal o eventual merge das duas será mais complexo e corre risco de haver bugs que pode prejudicar o progresso.

Master

É recomendado que a branch principal, master, esteja sempre funcional, ou seja tudo que estiver na master deve está funcionando. Muitos projetos adotam o padrão de ter a master com o progresso do projeto que já está funcionando e uma branch chamada development que contém o conteúdo em desenvolvimento e que não está 100%.

Boas práticas com ramificações

Branchs mortas

Branchs mortas são branchs que o conteúdo dela já foi unido com outra branch e não terá mais modificações na branch, logo a branch fica no repositório sem ser utilizada. É recomendado remover branchs que não terão mais utilidades no projeto.

Voltando para o passado

O git permite que o usuário volte para um determinado commit.

git checkout commit

O comando `git checkout <commit>` faz com que o usuário vá para a 'foto' do projeto.

```
git checkout <commit>;
git checkout -b <branch>
```

git revert e git reset

git revert

O comando `git revert <commit>` reverte alterações até o commit, criando um novo commit, logo ele desfaz modificações com um novo commit e mantém o histórico de commits anteriores.

git reset

O comando `git reset <commit>` 'descarta' todos os commits até o commit especificado. Portanto ele move o ponteiro do commit atual para o commit especificado, sem gerar um commit, assim perdendo a relação de commits entre o atual e o especificado.