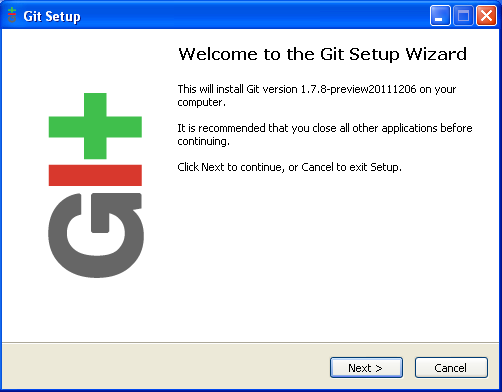
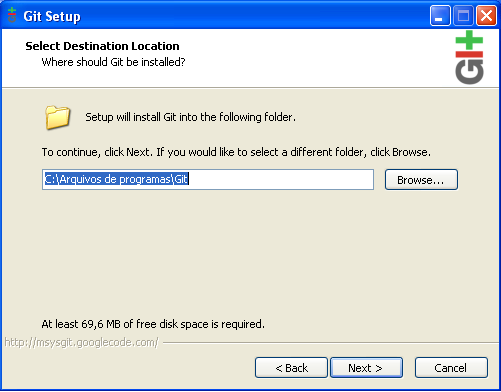
**Windows**

Para instalar o Git, inicialmente acessaremos a página <http://msysgit.github.io/> e baixaremos o instalador do msysgit. O instalador da versão mais recente, no momento da escrita deste curso, é o Git-1.9.5-preview20141217.exe. Uma vez baixado, basta executar o programa e seguir as instruções (caso haja uma tela de instalação que não esteja aqui, *pode* aceitar com as opções padrão).

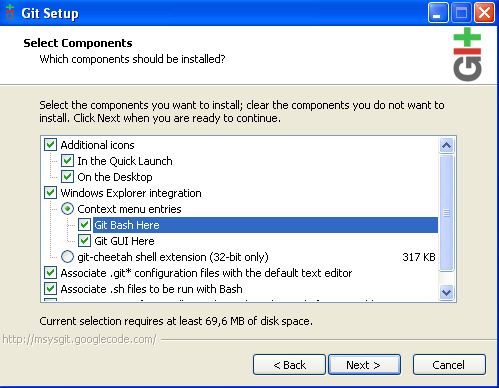
Na primeira tela de boas vindas, apenas clique em Next:



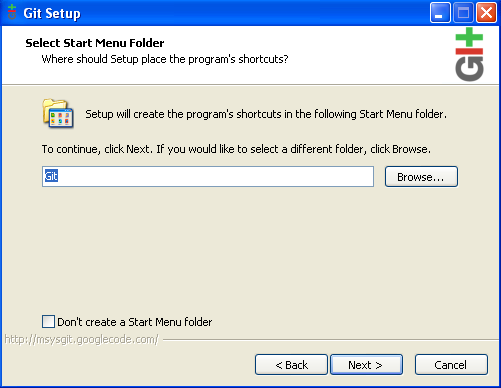
Na próxima tela, indique o diretório onde deseja instalar o Git em seu computador:



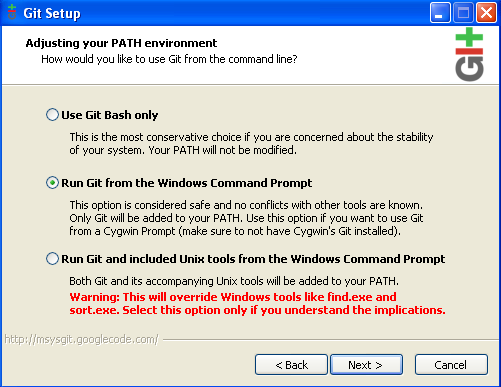
Em seguida, indique quais utilitários serão instalados junto do Git. Marque todas as opções, exceto "git-cheetah shell extension", como mostrado na imagem a seguir:



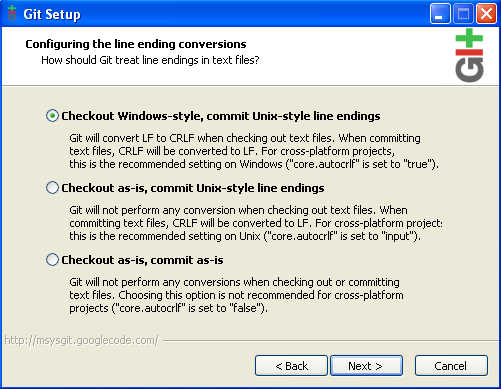
A seguir, defina o nome do menu do Git para o menu Iniciar do Windows:



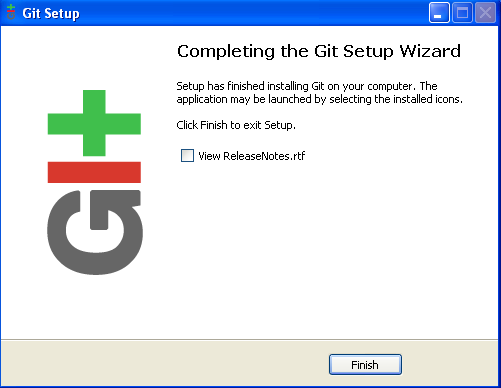
O msysgit instala por padrão em seu computador um programa próprio chamado Git Bash, onde é possível manipular os repositórios de arquivos que trabalharemos no futuro. Caso você prefira, é possível usar o próprio prompt do Windows. Basta, na próxima janela, marcar a opção "Run Git from the Windows Command Prompt", como indicado na imagem a seguir:



Na janela a seguir, indique que queremos que as quebras de linha sejam consideradas da mesma maneira, independente do sistema operacional trabalhado. Para isso, marque a primeira opção:



Nesse instante, a cópia dos arquivos será feita para o computador, e em seguida a mensagem de que a instalação foi finalizada aparecerá:



**MacOS**

Para instalar o Git no Mac OS X (Snow Leopard ou superior), há um instalador pronto. Basta baixá-lo em <http://code.google.com/p/git-osx-installer/downloads/list?can=3>. Na listagem, selecione o arquivo **git-1.7.7.3-intel-universal-snow-leopard.dmg**.

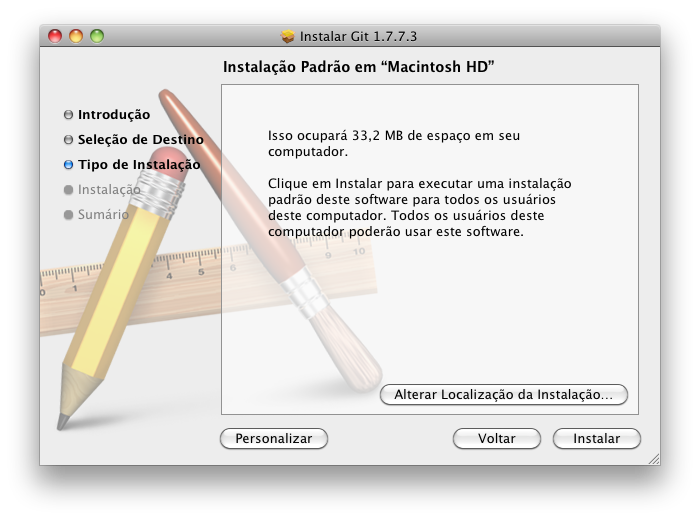
Após baixá-lo, basta clicar duas vezes para ter acesso ao pacote de instalação. Agora é só clicar duas vezes no arquivo "git-1.7.7.3-intel-universal-snow-leopard.pkg" (de acordo com a versão encontrada no momento da escrita) para iniciar o processo de instalação:

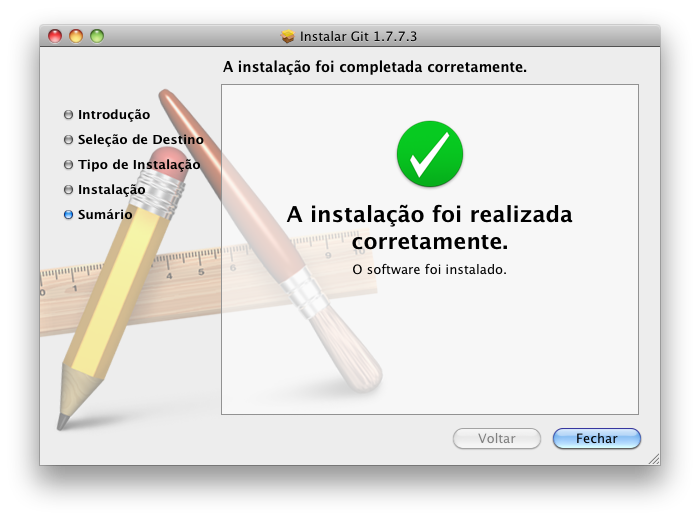


Na tela seguinte, selecione a opção "Instalar para todos os usuários desse computador" para continuar:

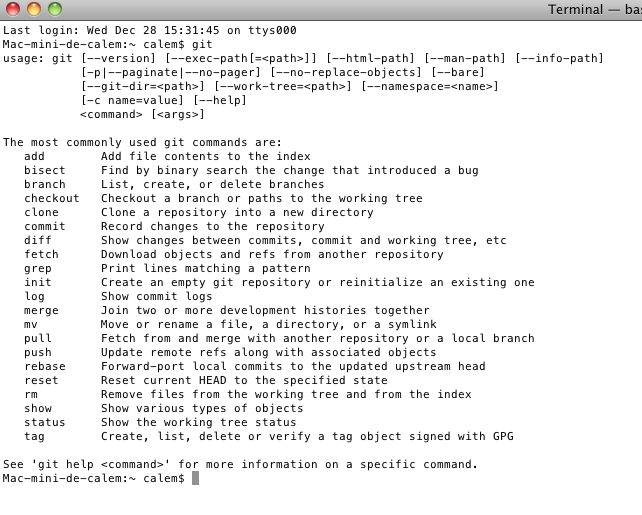


Depois é só prosseguir com a instalação padrão.





Para testar a instalação, abra a aplicação "Terminal" e digite o comando git. A saída deve ser similar à imagem:



Opcionalmente, é possível configurar o Terminal para completar os comandos do Git ao pressionarmos a tecla "tab", além de mostrar na linha de comando se a pasta atual está sendo rastreada pelo Git. Para isso é necessário adicionar as seguintes linhas ao arquivo de perfil do usuário para o prompt de comando, habitualmente encontrado na pasta home do usuário com o nome de .bash\_profile (ou .bashrc).

Adicione as seguintes linhas ao fim do arquivo:

if [ -f /usr/local/git/contrib/completion/git-completion.bash ]; then

. /usr/local/git/contrib/completion/git-completion.bash

fi

GIT\_PS1\_SHOWDIRTYSTATE=true

PS1='\u@\h:\w $(\_\_git\_ps1 "(%s)")\$ '

Ao iniciar uma nova janela ou aba do Terminal, as alterações estarão aplicadas. Caso queira aplicar as alterações imediatamente no Terminal digite source ~/.bash\_profile.

Obs.: Caso tenha efetuado a instalação do GIT via Homebrew (<http://brew.sh/>), alterar o endereço do arquivo **git-completion.bash** para **/usr/local/Cellar/git/\*/etc/bash\_completion.d/git-completion.bash**.

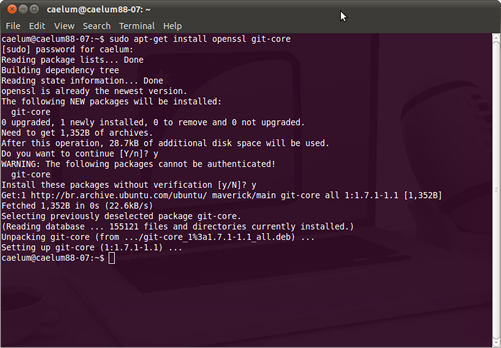
**Linux**

Em qualquer sistema Linux, podemos utilizar o gerenciador de pacotes da respectiva distribuição para instalar o Git. No Ubuntu e no Debian, por exemplo, basta instalar o pacote git-core para ter o Git instalado. No Fedora, o pacote git. E no Gentoo, o pacote dev-util/git.

Portanto, por exemplo, para fazer a instalação do Git no Ubuntu, basta executar o comando:

sudo apt-get install openssl git-core

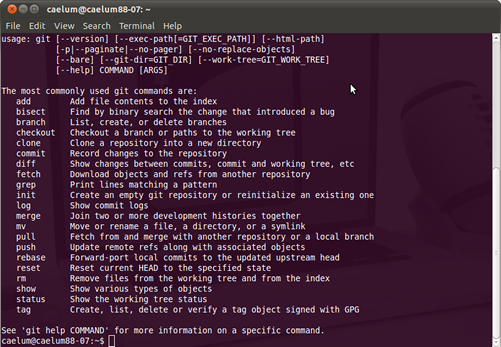
Siga as instruções do prompt de comando, primeiro confirmando a instalação dos pacotes e suas dependências, depois confirmando a instalação do pacote git-core, como demonstrado na imagem:



É possível testar se o Git foi corretamente instalado rodando o comando git no prompt de comando:

git

A saída deve ser uma mensagem de ajuda, similar à demonstrada na imagem:



Ao instalar o Git, um script de ajuda que completa os comandos do Git no terminal é instalado, juntamente com um script que demonstra na linha de comando se estamos em um diretório que é um projeto rastreado pelo Git. Para ativá-los, é necessário modificar o arquivo de perfil do prompt de comando. Normalmente esse arquivo encontra-se na pasta home do usuário e chama-se .bash\_profile (em alguns casos .bashrc ou .zshrc).

Caso nenhum dos arquivos citados exista, crie-o; caso exista, adicione as seguintes linhas ao final dele:

if [ -f /etc/bash\_completion.d/git ]; then

. /etc/bash\_completion.d/git

fi

GIT\_PS1\_SHOWDIRTYSTATE=true

PS1='\u@\h:\w $(\_\_git\_ps1 "(%s)")\$ '

Salve o arquivo. A partir de agora, caso estivermos em uma pasta rastreada pelo Git no prompt de comando, seremos informados. Para ativar as alterações imediatamente, execute o comando source informando o arquivo de perfil que alteramos:

source ~/.bash\_profile

**Criando conta no Github**

Agora que temos o Git instalado, podemos utilizar os serviços do Github. Primeiro precisamos criar uma conta. Para isso acessamos a página <https://github.com/plans>, escolhemos o tipo de conta que queremos criar e prosseguimos preenchendo os formulários pedidos.

Depois de criada nossa conta, precisamos gerar uma senha (chave de segurança) que será responsável por identificar nossa máquina quando fizermos as interações entre nosso projeto e o serviço do Github. Caso não exista essa configuração entre nossa máquina e a conta do Github, o acesso ao serviço pelos comandos do Git será negado.

O processo de criação da chave de segurança é similar em todos os sistemas operacionais: basta abrir o prompt de comando (no caso do Windows, é necessário abrir o Git Bash) e inserir o seguinte comando:

ssh-keygen -t rsa -C "seu\_email@provedor.com"

Lembre-se de substituir `seu\_email@provedor.compelo seu endereço real de email. A resposta do terminal vai perguntar em qual local do seu disco você quer salvar sua chave de segurança. Para evitar problemas, mantenha a opção padrão. Em algumas versões, pode ser necessário incluir também a opção-b 2048ou-b 4096` para indicar o tamanho da chave a ser gerada.

A seguir, será solicitada a entrada de uma senha para a chave de segurança. Caso o computador seja público ou compartilhado, é recomendado que sua chave esteja protegida por uma senha. Caso contrário, recomendamos que a senha seja ignorada, bastando pressionar Enter ao ser solicitada a senha e a confirmação de senha.

Ao término do processo você deve ver uma mensagem similar à seguinte:

Your identification has been saved in /Users/seu\_usuario/.ssh/id\_rsa.

Your public key has been saved in /Users/seu\_usuario/.ssh/id\_rsa.pub.

The key fingerprint is:

01:0f:f4:3b:ca:85:d6:17:a1:7d:f0:68:9d:f0:a2:db seu\_email@provedor.com

The key's randomart image is:

+--[ RSA 2048]----+

| .+ + |

| = o O . |

| = \* \* |

| o = + |

| o S . |

| o o = |

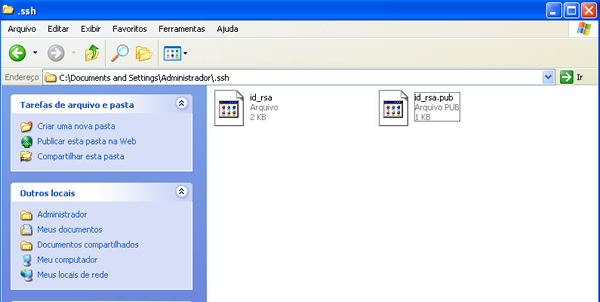
| o . E |

| |

| |

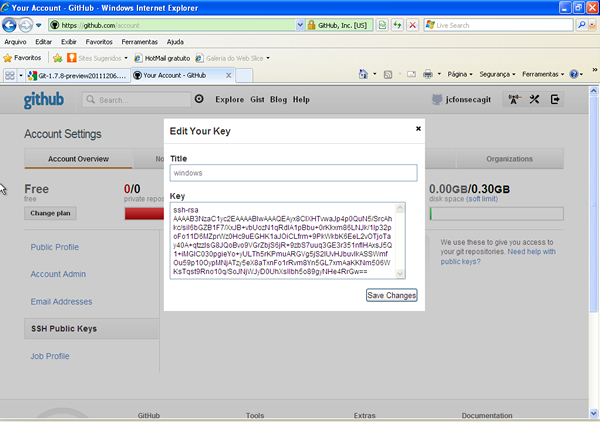
+-----------------+

Com isso, dois arquivos serão gerados: a sua chave privada e a sua chave pública, conforme pode ser visto na imagem a seguir:



Agora será necessário realizar o login no [Github](http://github.com/" \t "_blank) e seguir para as configurações do seu perfil. Siga "Settings", "SSH and GPG keys" e "New SSH key".

Insira na caixa de texto o conteúdo do seu arquivo id\_rsa.pub. O local exato do arquivo foi informado na saída do processo de criação da chave. Cole-o exatamente como ele está, sem adição de espaços ou quebras de linha. Para isso, abra-o no editor de textos simples do seu sistema operacional (TextEditor, Bloco de Notas, GEdit), como pode ser visto na figura a seguir:



Agora você tem o Git instalado em sua máquina, uma conta no Github e as configurações necessárias para utilizar o serviço com segurança.

**Copiando o projeto para sua máquina: o comando clone**

Iniciaremos o curso trabalhando em um repositório já existente. Para podermos fazer isso, nós temos que copiá-lo para nossa máquina. O repositório que vamos copiar está hospedado no Github. Para garantir que vamos adquirir todos os arquivos necessários, utilizaremos o comando git clone que, além de copiar os arquivos do projeto, traz todos os arquivos necessários para que o controle de versão dos arquivos continue sendo feito.

Abra o prompt de comando do computador e navegue até uma pasta que será a pasta "pai" do seu repositório local, por exemplo:

cd Documentos/Projetos

Lembre-se de que você pode escolher o local mais conveniente para você em seu computador. Agora vamos executar o comando que copiará o repositório para essa pasta:

git clone https://github.com/jcfonsecagit/repositorio.git

Esse comando **clone** criará uma pasta com o mesmo nome padrão do repositório (que no caso é "repositorio") e copiará para esse diretório todos os arquivos que estavam disponíveis nele.

Nosso projeto de exemplo contém páginas HTML. Para visualizar o projeto basta abrir o arquivo **index.html** em qualquer navegador (Chrome, Firefox, Internet Explorer, entre outros).

Você pode ver os arquivos do projeto navegando para a pasta "repositorio" e listando os arquivos no prompt de comando:

cd repositorio

ls

**Controle geral de versões: o comando tag**

Uma situação comum no desenvolvimento de um projeto de software é que, conforme ele vai evoluindo, vai ganhando novos releases e, consequentemente, novas versões. Com isso o software passa a ter a versão 1.0, 1.1, 2.0, 3.0 e assim por diante.

O Git nos permite, sempre que desejarmos, adicionarmos um marcador no repositório. Então, em determinado momento do projeto, é possível marcar o estado atual do repositório como a versão 1.0. Esse processo de adicionar um marcador, que habitualmente é o número da versão, é chamado de *tag*. É possível criar tags com qualquer nome, para que não fiquemos presos a um determinado padrão de nomenclatura de versões. Isso nos permite inclusive marcarmos o repositório como "v0.1a", "v0.1b" com a finalidade de definirmos versões preliminares, de acordo com a necessidade do projeto atual.

Mas como é possível visualizar as tags existentes num projeto? No Git, é possível listar todas as Tags existentes em um projeto através do comando tag:

git tag

Feito isso, como podemos fazer para voltar o repositório para alguma dessas versões? Por exemplo, queremos visualizar como o nosso projeto, junto com todos os seus arquivos html, estava na versão v0.1. Para isso, basta dizer ao git que queremos buscar a versão v0.1. Chamamos esse processo de **checkout** e podemos executá-lo com o comando a seguir:

git checkout v0.1

Abrindo a página html novamente, podemos verificar como ela estava na versão v0.1.

**Como saber o que mudou nos arquivos?**

E se entre a versão v0.2 e a v0.1 surgiu um bug? Como descobrir o que mudou entre elas para isolar onde pode estar o bug? As ferramentas de controle de versão possuem tarefas específicas para mostrar a diferença (diff) entre duas versões de qualquer elemento do projeto, podendo ser um arquivo, um diretorio etc. No nosso caso, queremos ver as diferenças entre a versão v0.2 e a v0.1, o que conseguimos com o Git através do comando git diff, onde indicamos entre quais versões desejamos visualizar as diferenças.

git diff v0.1 v0.2

Com isso, serão mostradas as linhas existentes numa versão que não aparecem na outra versão. Também serão mostradas as alterações efetuadas entre as versões selecionadas.

**Como descobrir quem realizou as alterações em um arquivo linha a linha?**

Dentro do projeto, é comum encontrar linhas de código cuja compreensão não é imediata. Como descobrir o motivo pelo qual uma linha foi alterada ou o motivo para que ela tenha sido implementada de uma determinada maneira? Se analisando apenas o contexto não é possível compreender o motivo, que tal falar com o responsável por aquela modificação?

O Git nos fornece uma funcionalidade onde podemos consultar quem foi o autor de cada linha de um arquivo através do comando git blame:

git blame index.html

Para sair do blame, basta apertar a tecla q

**Conclusão**

Como vimos através dessa apresentação simples de apenas alguns comandos disponíveis no Git, temos uma ferramenta que permite muito mais segurança no controle de alterações de arquivos de um projeto. A utilidade dessa ferramenta é indiscutível: com ela, é possível realizar com facilidade alguns controles que antes eram muito trabalhosos ou até mesmo impossíveis de se fazer quando trabalhamos simplesmente com pastas compartilhadas em uma máquina ou servidor.

Durante este curso, aprenderemos outros comandos que o Git nos fornece para solucionar diversos problemas desde o trabalho básico de criação de um repositório, commits, branches, conflitos, como trabalhar no repositório sem conexão com a internet, trabalhar em diferentes implementações simultaneamente num mesmo projeto, modificação de um arquivo por mais de uma pessoa simultaneamente no projeto e tratar os conflitos, entre outros.