Precisamos primeiro separar quem é quem, pois isso interfere no método de trabalho. Por questões de comoidade, daqui pra frente vou citar "controles de versão" e "sistemas de controle de versão" como VCS, que nada mais é que o acrônimo em inglês do segundo termo.

GIT é um VCS distribuído bla bla bla, acho que a parte da Wikipedia você já conhece, certo?

1. O que são e pra que servem as versões?
Sabe essas coisas de você baixar o Firefox 11 ou o Chrome 19? Basicamente, por trás do desenvolvimento, estão definindo marcos importantes na história daquele produto, são versões de um desenvolvimento. Uma mesma versão pode sofrer alterações significativas mas em quantidade insuficiente para caracterizar uma nova "Major Version", que é o número principal que vem depois do nome do Software. Nesse cenário, temos uma "Minor version". Uma Minor Version que figurou por muitíssimo tempo no cenário Web foi o Firefox 3.5.

Outras categorizações de versão incluem *patch*, que é um pacote de alterações normalmente voltado para correções de erros e falhas de segurança, *build* que é a contagem de "tentativas" de disponibilizar um software utilizável ou concluir uma tarefa de desenvolvimentot e *revision* que pode ser a busca minunciosa pelo código-fonte atrás de algo que você descobriu que pode ser uma falha ainda não explorada, por exemplo, ou a correção de uma regressão - bug corrigido que voltou.

2. Porque eu preciso salvar as minhas versões?
Podemos pegar como exemplo a última característica citada do tópico anterior: correção de regressão.

Uma vez que temos registrado **onde** o bug ocorreu e **como** ele foi corrigido, temos um importante ponto de partida que pode até mesmo resolver o nosso problema, talvez reaplicando a solução que foi sobrescrita por se achar não mais necessária. É possível retroceder em tudo que foi desenvolvido para realizar diversas análises - melhoria de código, de performance, de documentação - e até mesmo fazer comparativos de lógicas diferentes para confirmar se, realmente, o que estamos utilizando atualmente é melhor que uma solução que foi aplicada anteriormente.

Imagine, também, que lançamos uma nova versão de um software qualquer, que não teve boa aceitação da nova interface como, por exemplo, disposição dos botões, remoção de algum elemento ou a versão anterior era mais amigável para celulares e tablets. Novamente, podemos selecionar apenas estas características e aplicar à nossa versão atual.

3. Porque eu preciso de um sistema para controlar minhas versões?
Vários (vários mesmo) motivos, mas eu tenho um, muito simples, que vai convencer

qualquer um que ler este post: Você não precisa ficar criando pastinhas 1.0, 2.0, 3.0... - com cópias completas do sistema - para cada vez que concluir um desenvolvimento. Fica tudo centralizado, no mesmo diretório, indexado de forma otimizada, compactada e documentada.

4. E pra desenvolver em equipe?
Quando falamos de trabalho em equipe, é importante que todos os envolvidos trabalhem sobre a mesma versão para evitar diversas dores de cabeça no momento da união dos códigos. VCS's ajudam, muito, também, nesta tarefa. A mescla é feita linha-a-linha - quando possível - modificando somente o que for novidade e mantendo um registro completo desta modificação, como quando aconteceu e quem fez.
VCS's impedem que ocorra o clássico problema de conflito, quando versionamos no sistema de arquivos, de a próxima atualização substituir arquivos novos por arquivos antigos que não foram trabalhados. Exemplo

NetBoy trabalha no arquivo 1.
Evandro trabalha no arquivo 2.

Evandro tem uma cópia completa do sistema e atualiza - via pastas (ftp, que seja). Quando NetBoy terminar e enviar a cópia completa, estará enviando um arquivo B desatualizado, desfazendo o trabalho de Evandro.

Agora troque "no arquivo N" por "na linha N do arquivo A". O problema é um pouco mais delicado de se resolver. Um VCS trata tudo isso, fornecendo, inclusive, uma interface para tratamento de conflitos, quando um desenvolvedor tenta, a partir de uma versão desatualizada, alterar uma linha que já foi atualizada e não consta no repositório local.

5. O que são VCS's centralizados e distribuídos?
A diferença entre VCS's centralizados e distribuídos está na forma como você controla o repositório.

Para ter um VCS centralizado - CVS e Subversion, por exemplo - é necessário ter um servidor que atenderá as requisições de atualização. O repositório ficará amarrado àquele endereço de servidor e só será atualizado através do mesmo.

Já um VCS distribuído - GIT e Mercurial, por exemplo - leva uma cópia completa de si onde quer que seja despejado. Não é necessário estar online ou conectado a um servidor específico para adicionar uma nova versão ao sistema. Para atualizar sim, obviamente, mas qualquer espelho, sob qualquer endereço, pode servir de "atualizador" para o seu repositório. Até mesmo dois diretórios diferentes na sua máquina podem ser duas cópias do mesmo repositório onde uma atualiza a outra. É claro que isso não tem aplicação prática alguma.

6. E onde entra o GITHUB?

GITHUB se dispõe a ser mais um local onde há uma cópia completa do seu repositório, atuando como servidor centralizado de versões, mas sem essa real necessidade. Há, claro, a interface visual, a integração social, mas a grande sacada é estar disponível a qualquer hora e momento para um desenvolvedor que tenha acesso à Internet, sem a necessidade de configurar um servidor, abrir portas no firewall e ficar cuidando de DNS ou IP.