

Comandos básicos do Git

Git é um sistema de controle de versão distribuído, open source, rápido e eficiente. Desenvolvido inicialmente por Linus Torvalds, mesmo criador do kernel do Linux, começou a ganhar notariedade quando passou a ser utilizado como sistema de versionamento padrão para o projeto do kernel. Em 2008, com o lançamento do github, uma espécie de rede social para compartilhamento de códigos, o git deu um grande salto rumo a popularização.  
Vou abordar de forma rápida e resumida alguns comandos. Informações mais detalhadas podem ser encontradas na documentação do projeto ou vem vários tutoriais existentes pela web.

Primeiros Passos

Configurando informações sobre o autor dos commits:

|  |
| --- |
| git config --global user.name "Samuel"  git config --global user.email "samuel@samukapsilva.net" |

É possível alterar essas informações no arquivo ~/.gitconfig

Criando um repositório local:

|  |
| --- |
| cd meuprojeto  git init |

Para ter certeza que o repositório foi criado:

|  |
| --- |
| git status |

Áreas de Trabalho

O git possui 4 áreas de trabalho:  
1. O diretório .git que é o repositório contendo todos os arquivos versionados;  
2. Working Area que é um snapshot do .git dentro de um determinado momento no tempo;  
3. Stage que é um local temporário que armazena a referência para arquivos a serem versionados antes de serem commitados;  
4. Stash que também é um local temporário que pode armazenar e esconder arquivos que estão no Stage.

Adicionando arquivos novos ou modificados no Stage:

|  |
| --- |
| git add arquivo.txt  git add \*.py  git add . (para add todos os arquivos)  git add -i (para modo interativo. 1-5 ou 1,2,3,4 e -3 para retirar) |

Removendo arquivos não versionados do Stage:

|  |
| --- |
| git rm --cached arquivo.txt  git clean -fd (remove todos arquivos e diretórios) |

Removendo arquivos versionados e modificados do Stage:

|  |
| --- |
| git reset HEAD arquivo.txt  git reset HEAD (todos os arquivos) |

Desfazendo modificações de arquivos versionados no Stage:

|  |
| --- |
| git checkout -- arquivo.txt |

Trabalhando com o Stash:

|  |
| --- |
| git stash (Move todos os arquivos do Stage para o Stash)  git stash save "Mensagem" (Move todos os arquivos do Stage para o Stash e os identifica com uma mensagem)  git stash list  git stash apply (Recupera os arquivos do último Stash de volta para o Stage mantendo cópia no Stash)  git stash apply <ID> (Recupera os arquivos do Stash identificado pelo ID obtido pelo git stash list. Ex.: stash@{0})  git stash pop (Faz o mesmo que apply porém apaga os arquivos do Stash)  git stash drop <ID> (Apaga completamente o Stash)  git fsck --unreachable | grep commit (Recupera arquivos apagados do Stash) |

Commits

Apenas arquivos no Stage podem ser commitados.

|  |
| --- |
| git commit -m "Mensagem"  git commit -a -m "Mensagem" (commita também os arquivos versionados mesmo nao estando no Stage) |

Refazendo commit quando esquecer de adicionar um arquivo no Stage:

|  |
| --- |
| git add arquivo.txt  git commit -m "Mensagem" --amend |

O amend é destrutivo e só deve ser utilizado antes do commit ter sido enviado ao servidor remoto.

Voltando commits anteriores:

|  |
| --- |
| git revert -m 1 <commit sha> (Desfaz pushed merge)  git reset --hard HEAD~1 (volta ao último commit)  git reset --hard <commit sha>  git reset --soft HEAD~1 (volta ao último commit e mantém os últimos arquivos no Stage)  git reset --hard XXXXXXXXXXX (Volta para o commit com a hash XXXXXXXXXXX)  git reset --merge ORIG\_HEAD |

Recuperando commit apagado pelo git reset:

|  |
| --- |
| git reflog (Para visualizar os hashs)  git merge <hash> |

Logs

Visualizando logs:

|  |
| --- |
| git log  git log --stat (Mostra o que foi modificado em cada commit)  git log --graph (Mostra gráfico do log)  git log --pretty=oneline (Mostra os commits linha por linha)  git log --pretty=format:"%an %ad %h %s" (Exibe o autor, data, sha1 abreviado e texto do commit)  git log --since=30.minutes ou 1.hour ou 2.hours (Exibe commits dos últimos 30 minutos, 1h ou 2h)  git log --since=10.hours --until=2.hours (Exibe commits entre as últimas 10h e últimas 2h)  git log --before="2010-12-25" (Exibe commits antes do dia 25/12/2010)  git reflog (Mostra commits apagados pelo git reset) |

Branches

Cada branch deve ter uma única funcionalidade. É recomendado criar um novo branch a partir do master e aplicar os merges nele para efeito de simulação.

|  |
| --- |
| git branch (Lista os branches)  git branch -a (Mostra também os branches do repositório remoto)  git branch -d novobranch (Apaga o branch)  git branch -D novobranch (Força a remoção do branch)  git checkout -b novobranch (Cria um branch contendo os mesmos commits do branch de origem)  git checkout -b novobranch origin/outrobranch (Cria novobranch a partir do outrobranch no repositório remoto)  git checkout -b [branch, tag, sha1]  git checkout -b <branch> v1.0 (Cria um branch a partir da tag v1.0)  git checkout master (Retorna ao branch master)  git rebase master (Atualiza um branch com o que há de novo no master)  git merge novobranch (Faz um merge do que foi feito em novobranch)  git merge novobranch --squash (Permite definir uma nova mensagem em vez das mensagens de todos os commits do novobranch) |

Conflitos

Quanto mais tempo demorar para atualizar um branch a partir do master (git rebase), maior será a chance de haver conflitos depois.  
O rebase é destrutivo, se estiver trabalhando em um servidor remoto deve usar o merge.

|  |
| --- |
| git rebase --skip (Perde o arquivo novo)  git rebase --abort (Cancela o rebase)  git rebase --continue (Para continuar após lidar com o conflito manualmente) |

Repositórios

Clonando repositórios:

|  |
| --- |
| git clone repo1 repo2 (Clona um repositório e add o repo1 como orign no repo2)  git remote show origin (Origin é uma convenção para o primeiro remote)  git push origin (Envia o commit local para o repositório remoto)  git push origin outrobranch (O mesmo acima mas para um determinado branch)  git remote add origin repo (Adiciona um repositório como remoto)  git pull (Atualiza a partir do repositório remoto)  git pull origin outrobranch (O mesmo acima mas a partir de um determinado branch)  git remote rm origin (Remove o repositório remoto) |

Trabalhando como repositórios remotos:  
Antes de dar um git push, dar um git fecth e um git rebase para não criar conflitos para outros usuários.

|  |
| --- |
| git init --bare (Cria um repositório sem área de trabalho)  git fetch origin (Puxa novos commits do repositório remoto)  git fetch remote <branch> (Puxa novos commits do repositório remoto para o branch)  git push origin <branch> (Envia o que está no branch atual para o branch no repositório remoto)  git push origin v1.0 (Envia a tag v1.0)  git pull (Atualiza o repositório local a partir do remoto. Similar a usar "fecth" + "merge")  git pull origin <branch> (Atualiza o branch local a partir do branch remoto) |

Github

Criando seu próprio projeto:  
Crie um projeto pelo site do github. Em seguida, na máquina local, crie um par de chaves pública e privada, copie e cole no campo apropriado no github.

|  |
| --- |
| ssh-keygen -t rsa |

Depois copiar o conteudo de ~/.ssh/id\_rsa.pub e colar na página do github.

Fazendo um fork de um projeto:  
Faça um fork de um repositório, um clone para sua máquina, altere o código, commit e no site clique no link “pull request”. O dono do repositório original deve adicionar a URL do repositório fork com git remote add usuario urlfork. Depois executar um git fecth para trazer os branches do fork. Usar git diff usuario/ para ver as alterações. Para aceitar, git merge(resolver conflitos caso apareça), criar um novo commit e enviar com o git push. O usuário que fez o fork deve executar o mesmo procedimentos para manter o fork sincronizado com o repositório original.

Patches

Trabalhando com patches:

|  |
| --- |
| git format-patch <branch> --stdout > patch.diff (Cria um patch)  git am patch.diff (Aplica o patch) |

Tags

Uma tag é utilizada para criar uma versão de lançamento.

|  |
| --- |
| git tag v1.0 (Cria a tag v1.0)  git push origin v1.0 (Envia a tag v1.0)  git push --tags (Envia todas as tags)  git checkout -b <branch> v1.0 (Cria um branch a partir da tag v1.0) |

git-svn

Lidando com svn:

|  |
| --- |
| git svn clone svn://repo (Clona um repositorio svn)  git svn clone -r10:HEAD URL NOME (clone de um intervalo de revisões svn)  git svn dcommit (Envia commit para o repositório svn)  git svn fecth (Atualiza a partir do repositório svn) |

Links

<http://git-scm.com/>  
<http://www.kernel.org/pub/software/scm/git/docs/>