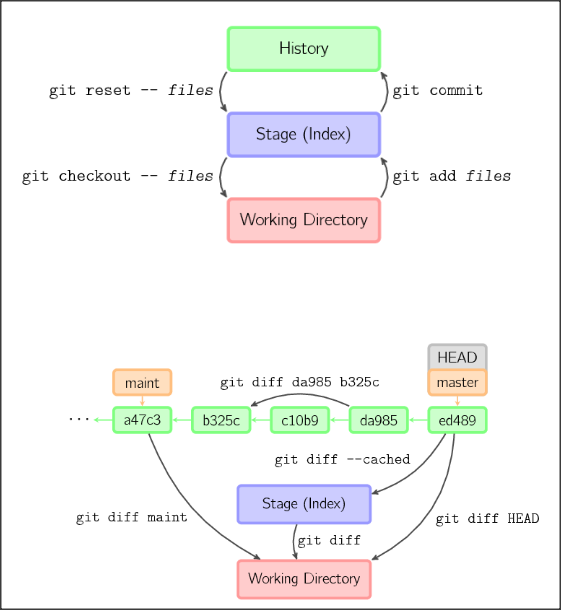
# Git installation

http://wikis.rim.net/display/SVVMCAT/Git+Setup#

# Git chart

****

**Check version**

$git –version

Git version 1.7.4

**Configuration**

Config file is under home directory (.gitconfig) global

System /etc/gitconfig

Change global setting(current user)

$git config --global user.name “First Last”

$git config --global user.email [firstlast@123.com](mailto:firstlast@123.com)

$git config --global color.ui true

Change system setting (all users) user must have administrator privilege

$sudo git config --system alias.st status

$sudo git config –system alias.ci commit

Edit config file

$cd /path/to/my/workspace/demo

$git config –e

$git config –e –global

$git config –e –system

Read value

$git config <section>.<key>

[core]’

Bare = false

$git config core.bare

False

Write value

$git config <section>.<key> <value>

$git config a.b something

[a]

B = something

$git config x.y.z others

[x “y”]

Z = others

**Use git config to change other INI file(config)**

$GIT\_CONFIG=test.ini git config a.b.c.d “hello, world”

Read

$GIT\_CONFIG=test.ini git config a.b.c.d

Hello, world

Delete setting

$git config –unset –global user.name

**Git initialization(init)**

$cd /path/to/my/workspace

$mkdir demo

$cd demo

$git init

Initializaed empty Git repository in /path/to/my/workspace/demo/.git/

If git version is higher than 1.6.5

$cd /path/to/my/workspace

$git init demo

Initializaed empty Git repository in /path/to/my/workspace/demo/.git/

$cd demo

**Add file to be commited(add)**

$git add <file name>

**Commit file to version control system(commit)**

$git commit –m “initialized”

*-m*

Add updates comments

Allow submit empty file

--allow-empty

$git commit –allow-empty

--*amend*

Fix the submission info

*--reset-author*

Fix author id, date

**Search files(grep)**

$git grep “search content”

**Check version status(status)**

Trace search path

$strace –e ‘trace=file’ git status

$git status

-s

Use simple format

$git status –s

M filename

M means modified. If M is displayed in col 1, this file is modified between Repository and Stage(2 and 3).

If M is displayed in col 2, this file is modified between work copy and stage(1 and 2).

Repository (3)

Stage (2)

Work Copy (1)

**Find .git location**

Find .git folder location

$cd /path/to/my/workspace/demo/

$mkdir –p a/b/c

$cd /path/to/my/workspace/demo/a/b/c

$git rev-parse –show-toplevel

/path/to/my/workspace/demo

$git rev-parse –show-prefix

a/b/c

$git rev-parse –show-cdup

../../../

**Log info(log)**

–pretty=fuller

Use full format

$git log –pretty=fuller

--pretty=oneline

Use simple format

$git log –pretty=oneline

–stat

Display file change history

$cd /path/to/my/workspace/demo

$git log –stat

Commit 40 bit commit id

Author: xxxx xxx<email>

Date:

**Compare file(diff)**

$git diff

Diff –git a/xxxxx.filename b/xxxxx.filename

Index

--- a/filename

+++ b/filename

HEAD

$git diff HEAD

Stage (2)

Work Copy (1)

Repository (3)

No paramter

$git diff

Compare the differences between work copy and stage (1 and 2)

HEAD

$git diff HEAD

Compare the differences between work copy and repository (1 and 3)

--cached, --staged

$git diff –cached or –staged

Compare the differences between stage and repository(2 and 3)

**Check reference (show)**

$git show-ref

Commit id refs/heads/master

Commit id refs/remotes/origin/HEAD

Commit id refs/remotes/origin/master

Commit id refs/tags/A

Refs/heads are branches

Refs/remotes are remote branches local reference.

Refs/tags are milestone

**Withdraw the changes (checkout)**

$git checkout – welcome.txt

$git checkout . (all files)

All change in work copy will be gone.

**Clean up file(clean)**

$cd /path/to/my/workspace/demo

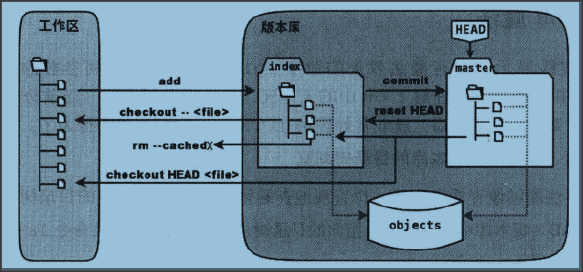
$git clean –fd

$git checkout .

Clean up the files in work copy which haven’t been added to stage

Replace the rest of files with stage.

**Unstage file(reset or checkout)**



**Git add(add file from work copy to stage)**

**Git checkout --<file> (work copy will be replaced by stage)**

**Git checkout HEAD. Or Git checkout HEAD <file> (work copy and stage file will be replaced by repository – HEAD)**

**Git commit(submit file from stage to repository - HEAD)**

**Git reset HEAD(stage will be replaced by repository –HEAD)**

**Git rm –cached <file> (the file will be deleted from stage)**

**Clone repository (Clone)**

$git clone

Copy repository A to another repository B. Use “push” and “pull” to sync up the changes between A and B

A clone B

A push to B

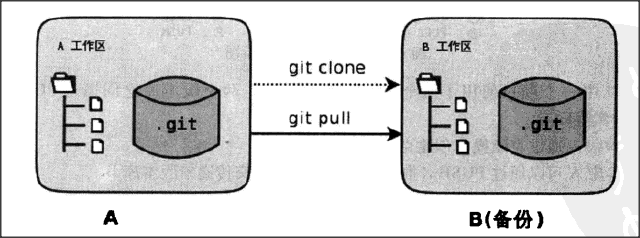
A pull from B

B pull from A

B push to A

$git clone <repository> <directory>

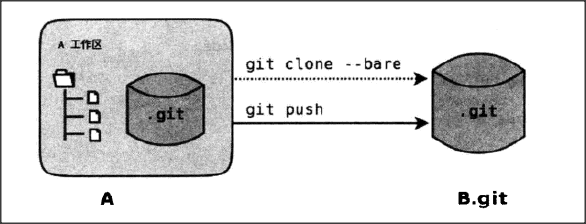
Clone <repository> HEAD version to <directory> folder. <directory> is a work copy of <repository>. The file objects is under .git folder.



*--bare*

$ git clone –bare <repository> <directory.git>

Clone <repository> HEAD version to directory.git. This will only generate file version info (only file under .git folder in above command). No work copy.



*--mirror*

$git clone –mirror <repository> <directory.git>

Same as –bare command, but this will remember the remote file version. So you can use git fetch to sync up with remote file version (only file version, no work copy).

**Push/Pull file (Push, Pull)**

$git push [<remote-repos>] [<refspec>]

$git pull [<remote-repos>] [<refspec>]

<remote-repos> the remote repository address or name

<refspec> reference to file version

# GitHub Tutorial

http://www.worldhello.net/gotgithub/03-project-hosting/020-repo-operation.html

**Account setup**

Sign up git hub account (use rim credential for github.rim.net)

Add ssh key to your profile.

Command:

Ssh-keygen –c “emailaddress” –f ~/.ssh

This will generate two key(public and private) under .ssh folder. Id\_rsa.pub is public key, id\_rsa is private key. Open public key in a text editor tool and copy and paste into your github account profile.

**Create repository**

**New project:**

click new repository. Type project name, description and homepage URL.

If this is a new repository, we can do a clone of this empty project then commit changes, and push to github

$git clone [git@github.com:<username>/<project>.git](mailto:git@github.com:%3cusername%3e/%3cproject%3e.git)

Create readme.md file

$git add readme.md

$git commit –m “readme for this project”.

$git push origin master

**Start with an existing project**

In local computer

$mkdir <project>

$cd <project>

$git init

Add readme.md file

$git add README.md

$git commit –m “README for this project”

Add remote “origin”

$git remote add origin [git@github.com:<username>/<prjoect>.git](mailto:git@github.com:%3cusername%3e/%3cprjoect%3e.git)

$git push –u origin master

**Create Branch**

Git branch was a reference which saved under ./git/refs/heads/. Content is the branch commit id

For example, the default master -> .git/refs/heads/master

How to create a branch in github

1. Create new branch in local
2. Push the new branch from local to github

Create new branch mybranch1

$git checkout –b mybranch1

$touch hello1

$git add hello1

$git commit –m “add hello1 from mark”

$git push –u origin mybranch1

Go to github, you will be able to see a new branch “mybranch1” on the webpage.

**Setup default branch**

Click “Admin”, change default branch in “Options”

After change the default branch, if you clone repo from github, the default branch will be pointing to default branch.

$ git branch -r

origin/HEAD -> origin/mybranch1

origin/master

origin/mybranch1

HEAD and mybranch1 pointing to the same location

$ git ls-remote

From git@github.com:gotgithub/helloworld.git

f46a28484adb6c1b4830eb4df582325c740e9d6c HEAD

e1e52d99fa71fd6f606903efa9da04fd0055fca9 refs/heads/master

f46a28484adb6c1b4830eb4df582325c740e9d6c refs/heads/mybranch1

**Delete a branch**

$ git branch -d mybranch1

error: Cannot delete the branch 'mybranch1' which you are currently on.

$ git checkout master

$ git branch

\* master

mybranch1

Switch to master branch.

$ git branch -D mybranch1

Deleted branch mybranch1 (was f46a284).

By default, git doesn’t allow to delete a branch before it merges. Use –D to ignore this warning.

**Delete branch from github**

Have to use git push instead of git branch to delete a branch from github

$ git push origin :mybranch1

remote: error: refusing to delete the current branch: refs/heads/mybranch1

To git@github.com:gotgithub/helloworld.git

! [remote rejected] mybranch1 (deletion of the current branch prohibited)

error: failed to push some refs to 'git@github.com:gotgithub/helloworld.git'

change default branch to master

$ git push origin :mybranch1

To git@github.com:gotgithub/helloworld.git

- [deleted] mybranch1

list remote branches.

$git ls-remote [git@github.com:<username>/<project>.git](mailto:git@github.com:%3cusername%3e/%3cproject%3e.git) list all remote branch

$ git ls-remote git@github.com:gotgithub/helloworld.git

From git@github.com:gotgithub/helloworld.git

e1e52d99fa71fd6f606903efa9da04fd0055fca9 HEAD

e1e52d99fa71fd6f606903efa9da04fd0055fca9 refs/heads/master

**Manage Tags**

Tag is very similar to branch but it is saved under .git/refs/tags/

Light way, create a tag refering to a <commit>

$git tag <tagname> [<commit>]

With descriptions, create a tag object with descriptions.

$git tag –a <tagname> [<commit>]

With signatures.

$git tag –s <tagname> [<commit>]

Example:

Add a file and commit it

$ touch hello1

$ git add hello1

$ git commit -m "add hello1 for mark."

Create mytag1, mytag2 and mytag3

$ git tag -m "Tag on initial commit" mytag1 HEAD^

$ git tag -m "Tag on new commit" mytag2

$ git tag mytag3

Check the new tags

$ git tag -l -n1

mytag1 Tag on initial commit

mytag2 Tag on new commit

mytag3 add hello1 for mark.

Push tags to github

$ git push origin refs/tags/\*

Counting objects: 6, done.

Delta compression using up to 2 threads.

Compressing objects: 100% (4/4), done.

Writing objects: 100% (5/5), 548 bytes, done.

Total 5 (delta 0), reused 0 (delta 0)

To git@github.com:gotgithub/helloworld.git

\* [new tag] mytag1 -> mytag1

\* [new tag] mytag2 -> mytag2

\* [new tag] mytag3 -> mytag3

**Delete tags from local**

$ git tag -d mytag3

Deleted tag 'mytag3' (was c71231c)

**Delete tags from github**

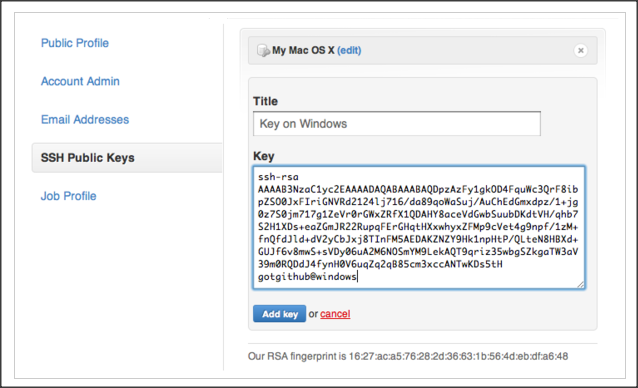
$ git push origin :mytag3

To git@github.com:gotgithub/helloworld.git

[deleted] mytag3

**SSH key management**

User might access github from different computer, we need to add more than one public key, go to “edit profile” -> “SSH public keys”



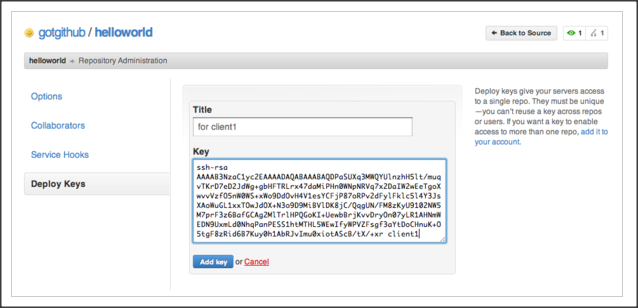
After you add public key to github, you can use the same user account to access github from different location. You can use ssh –T to test the authentication and get your account name.

$ ssh -T git@github.com

Hi gotgithub! You've successfully authenticated, but GitHub does not provide shell access.

**Project permission management**

Login into github with your account, go to the <project> repository you want to manage, choose “Deploy Keys” to set the project level privilege.



Once you set project github SSH, you can use ssh –I to display project id instead of user id.

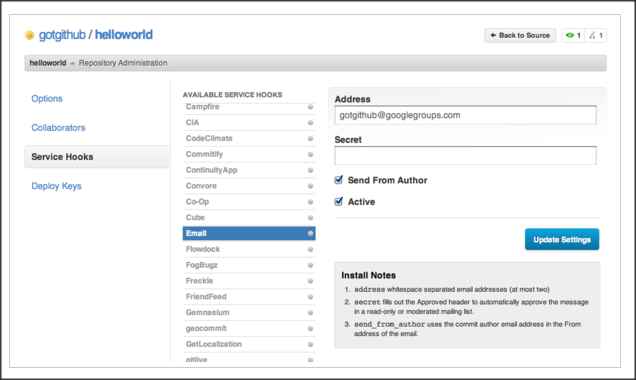
$ ssh -i ~/.ssh/deploy-key -T git@github.com

Hi gotgithub/helloworld! You've successfully authenticated, but GitHub does not provide shell access.

**Service Hook**

**Email notification**

Add email address to “email” under “Service Hooks”



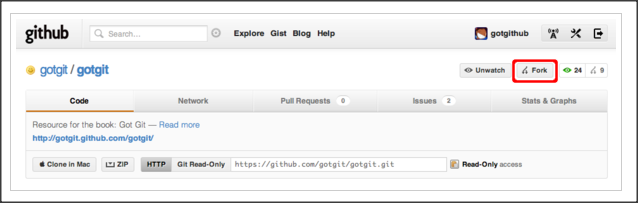
When new commit is pushed to github, you will be notified by email

**Collaboration**

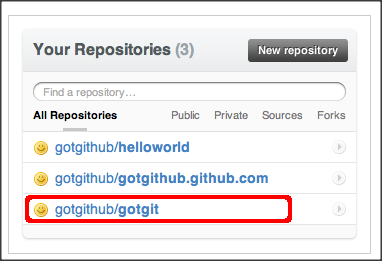
**Fork + Pull**

In github, you don’t have to request permission to work on the project, you can work on your own repo by using “fork”

Go to the project you want to work on, click fork, you will get a copy of project in your own repo



Go to your own repo, you will be able to see the project you forked under your project list.



The new repo will be same as the origin project except that it doesn’t have issue tracker. And you will be able to see the original project path.



**Clone project**

Clone the project of your own repo

$ git clone git@github.com:gotgithub/gotgit.git

$ cd gotgit

Set user name and email (this will be used to track commit history)

$git config user.name “fwang”

$git config user.email [fwang@rim.com](mailto:fwang@rim.com)

Add a new file

$touch newfile

$git add newfile

$git commit –m “add newfile”

Push the change to github

$git push

Go to your own repo on github, you should see a commit under master branch.

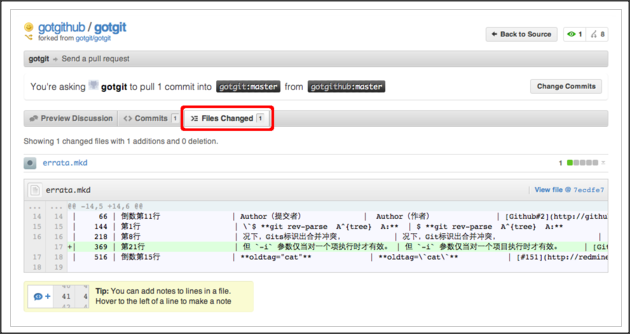
**Pull request**

After you commit the changes, you want to let the project owner to merge the changes. Github provides “pull request” to allow you send a pull request to project owner.

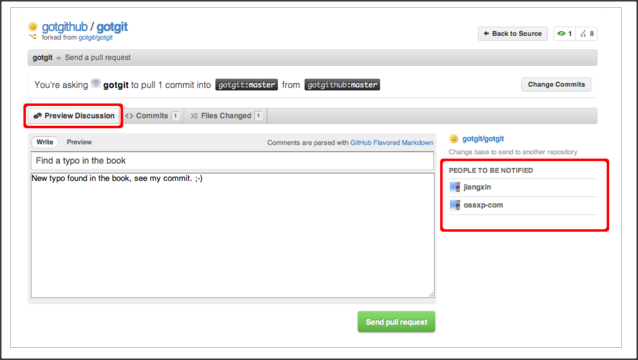
Click “pull request” -> “commits” to check what you can commit



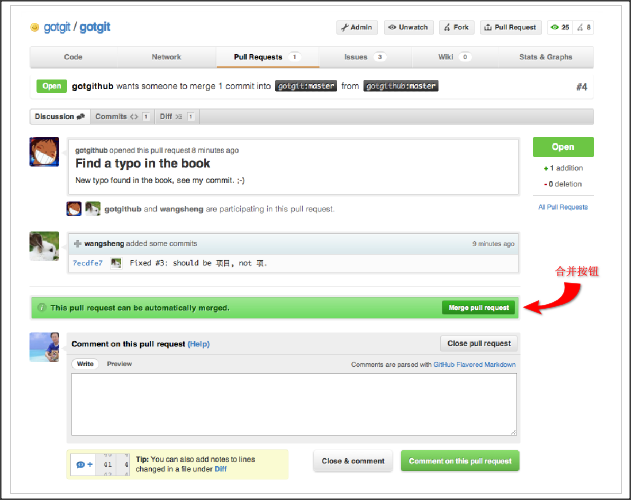
Click the “file changed” to check what has been changed



Click preview discussion to fill up the title, content and finish pull request creation.



An email will be sent to project owner. The project owner can click “merge pull request” to finish the merge.



If the merge can be completed automatically, github will display a dialog to let project owner to confirm the merge.



After this step, the merge is done.



**Merge manually**

How the merge log looks like

$ git clone git@github.com:gotgit/gotgit.git

$ cd gotgit

$ git log --graph -3

\* commit 6c1f1ee152629fd2f8d00ebe92c27a32d068d756

**|\ Merge: 00c6c4b 7ecdfe7**

| | Author: OpenSourceXpress <worldhello.net@gmail.com>

| | Date: Tue Aug 16 01:23:47 2011 -0700

| |

| | Merge pull request #4 from gotgithub/master

| |

| | Find a typo in the book

| |

| \* commit 7ecdfe7451412cfb2e65bb47c12cf2162e21c841

|/ Author: Wang Sheng <wangsheng@ossxp.com>

| Date: Tue Aug 16 10:17:53 2011 +0800

|

| Fixed #3: should be 项目, not 项.

|

\* commit 00c6c4bfab9824bd967440902ce87440f9e87852

| Author: Jiang Xin <worldhello.net@gmail.com>

| Date: Wed Aug 3 11:50:31 2011 +0800

|

| Change font color for stronger text from red to brown.

Look at the highlighted line in red, it does tell you that github did a auto merge for you, but in many cases, github cannot merge commit automatically, how are we gonna do about this?

Add a new remote for the pull request version repo

For example, received a pull request from user gotgithub,

$ git remote add gotgithub https://github.com/gotgithub/gotgit.git

Now we have two remote in repo, “gotgithub” just got created a moment ago, “origin” is the default when you clone the project to your machine.

$ git remote -v

gotgithub https://github.com/gotgithub/gotgit.git (fetch)

gotgithub https://github.com/gotgithub/gotgit.git (push)

origin git@github.com:gotgit/gotgit.git (fetch)

origin git@github.com:gotgit/gotgit.git (push)

Get all changes from gotgithub branch.

$ git fetch gotgithub

From https://github.com/gotgithub/gotgit

\* [new branch] gh-pages -> gotgithub/gh-pages

\* [new branch] master -> gotgithub/master

Now we got couple of remote branches

$ git branch -a

\* master

remotes/gotgithub/gh-pages

remotes/gotgithub/master

remotes/origin/HEAD -> origin/master

remotes/origin/gh-pages

remotes/origin/master

Merge master branch of gotgithub to your current branch

$ git merge gotgithub/master

Updating 00c6c4b..7ecdfe7

Fast-forward

errata.mkd | 1 +

1 files changed, 1 insertions(+), 0 deletions(-)

Check the commit log again.

$ git log --graph -2

\* commit 7ecdfe7451412cfb2e65bb47c12cf2162e21c841

| Author: Wang Sheng <wangsheng@ossxp.com>

| Date: Tue Aug 16 10:17:53 2011 +0800

|

| Fixed #3: should be 项目, not 项.

|

\* commit 00c6c4bfab9824bd967440902ce87440f9e87852

| Author: Jiang Xin <worldhello.net@gmail.com>

| Date: Wed Aug 3 11:50:31 2011 +0800

|

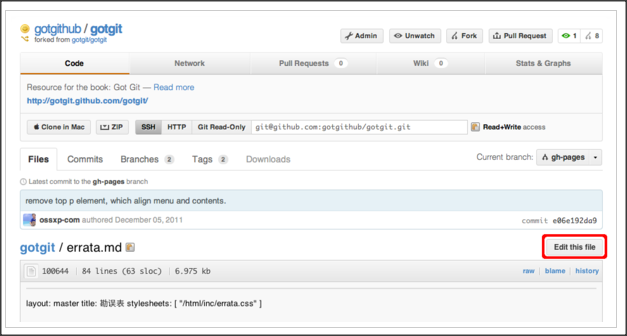
| Change font color for stronger text from red to brown.

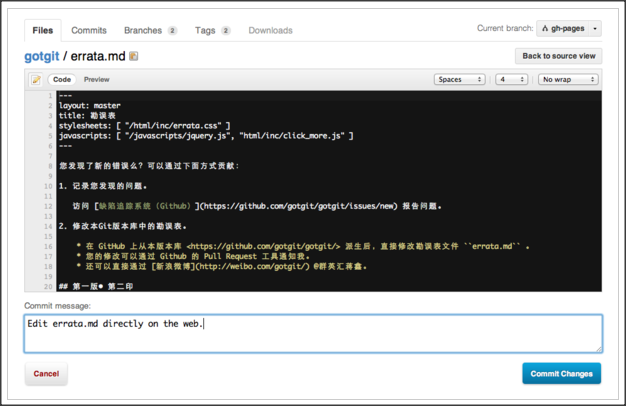
There is no auto merge this time. Push the change to github

$ git push

**Edit online**

If this is a easy quick change, you can edit the file online

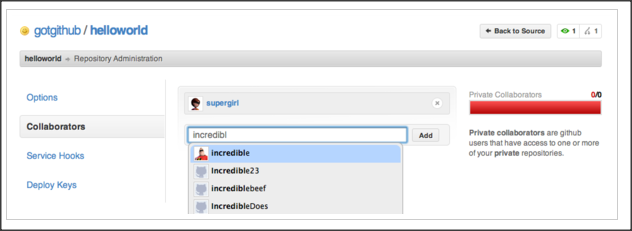




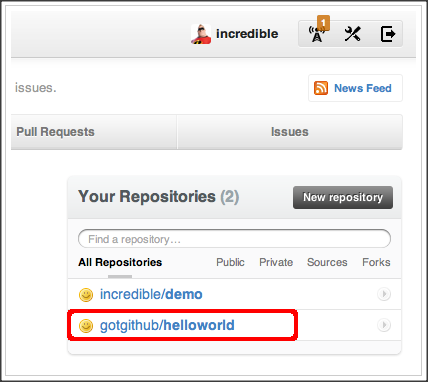
**Share repo**

**How to share repo across developers**

Go to the project admin tab, click “Collaborators”, you can add new developers to the project.



After the user is added to the project, you should be able to see the project in repository list



What’s the difference between git and other SVN tool

In traditional svn tool, users all access to same repo, but for git, user can all access to the centurial repo or they can clone the central repo to their own work space.

For example, if you add two user to the same project helloworld.git

User gotgithub clones the project

gotgithub$ git clone https://gotgithub@github.com/gotgithub/helloworld.git

User supergial clones the project

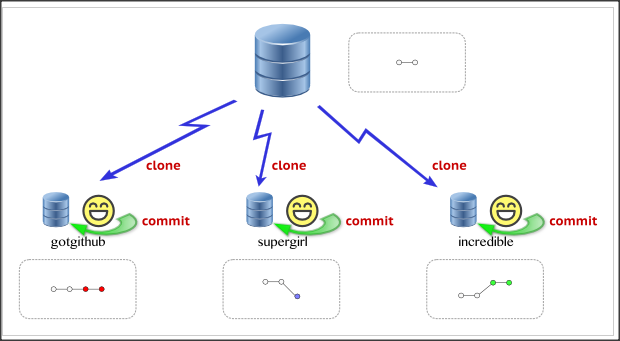
supergirl$ git clone https://supergirl@github.com/gotgithub/helloworld.git

User incredible clones the project

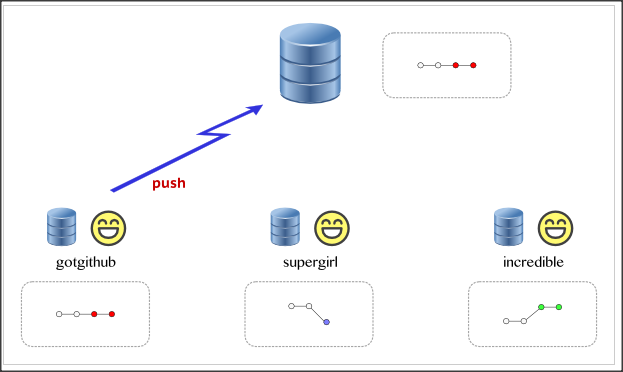
incredible$ git clone https://incredible@github.com/gotgithub/helloworld.git

In traditional svn tools, the commit are all saved in unique repo. Commit are submitted in sequence order. When you submit your changes, you must have connections to svn server, otherwise you will fail to submit the changes.

For git, all commits happen in local repo, only when you try to push or pull changes from origin repo.



From the image above, all there users are able to commit files to their local repo. When user gotgithub trys to push the change to server, it will be executed smoothly.



After gotgithub push the changes to central server, the other users have to pull the changes to local repo before they try to push changes to server.

supergirl$ git push

To https://supergirl@github.com/gotgithub/helloworld.git

! [rejected] master -> master (non-fast-forward)

error: failed to push some refs to 'https://supergirl@github.com/gotgithub/helloworld.git'

To prevent you from losing history, non-fast-forward updates were rejected

Merge the remote changes (e.g. 'git pull') before pushing again. See the

'Note about fast-forwards' section of 'git push --help' for details.

Github allows you to force push the changes to server by use git push –f. but this is not recommended strongly, since this will replace all previous changes with current change. The right way to do is to get all file from central repo and merge in your local repo then push to the server.

supergirl$ git fetch

supergirl$ git merge

or you can use git pull = git fetch + git merge

supergirl$ git pull

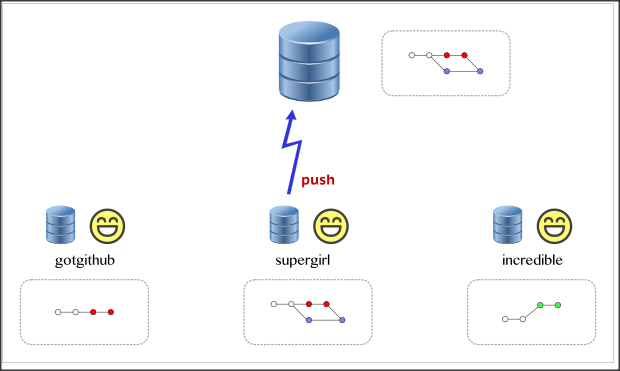
if conflicts exist, all conflict files will be saved in stage. You can use the command to load GUI tool to resolve conflicts

supergirl$ git mergetool

if the merge doesn’t go well, you can use git reset –-hard to go back to the state before merge starts.

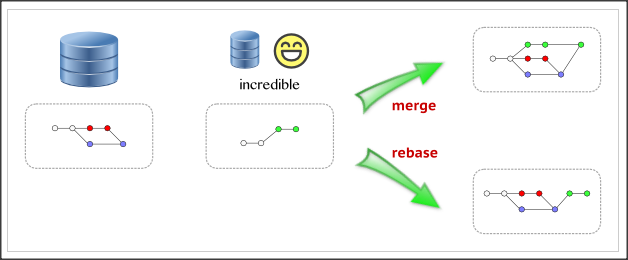
After the merge is done, push change to server

supergirl$ git push



**Merge or rebase**

Sometimes merge is not the ideal solution to code merge, it is going to have a hard time to do the code review or merge multiple changes into central repo. In certain cases, we can use rebase to handle the issues.



From the chart above, you can see the difference of merge and rebase.

Get the latest changes from server to local repo

incredible$ git fetch origin

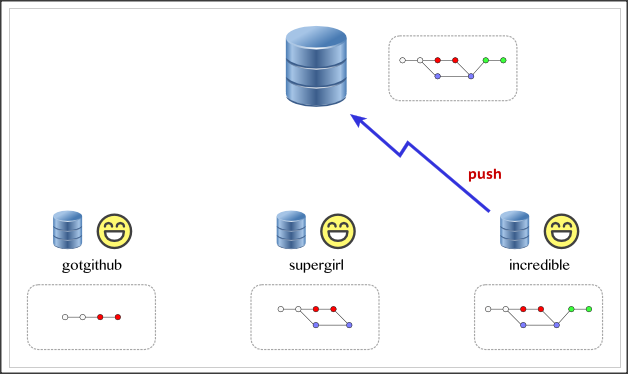
Rebase the changes, rebase the local master branch to new remote branch.

incredible$ git rebase origin/master

Push the changes to central repo.

incredible$ git push

After the push, the repo will be like the chart below



If you want to use one command to do the rebase, you can use git pull –--rebase to replace git merge

$ git pull --rebase

If you want to make rebase as the default value every time you use git pull command, you can set the config file by calling

$ git config branch.master.rebase true