Git common command tutorial

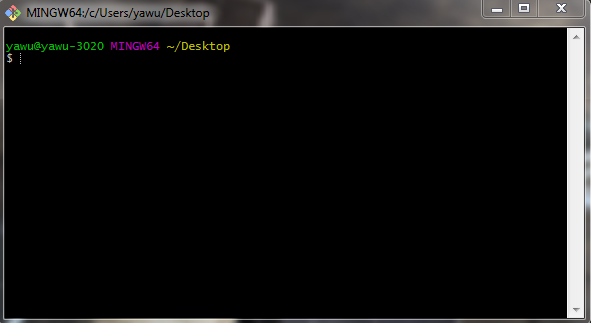
# Git introduction

Git is currently the most popular version control system. What is a version control system? For example, if you have written a document and named it "Document 01", then you add some content, named "Document 02". Later, you find that the document has been changed in a mess. I want to go back and change it again. The version control system will it helps us to fall back directly to "Document 01" instead of manually deleting the new content.

# Install Git for Windows

Download: <http://msysgit.github.io/> , install as default.

After the installation is complete, right click on the mouse and select **Git Bash Here**. The following window appears and the installation is successful.



After the installation is complete, type it at the command line:

$ git config --global user.name "Name"

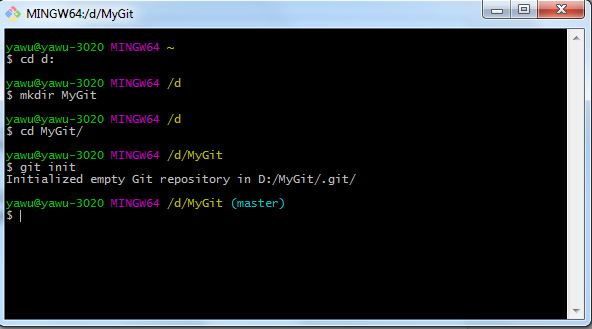
$ git config --global user.email "email@example.com"

This way, when you submit the code, others will know who submitted it.

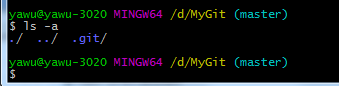
# Initialize a repository

First create a folder **MyGit**, right click after entering, select **Git Bash Here**, and type **git init**.

Or open **Git Bash** directly, using the command line.



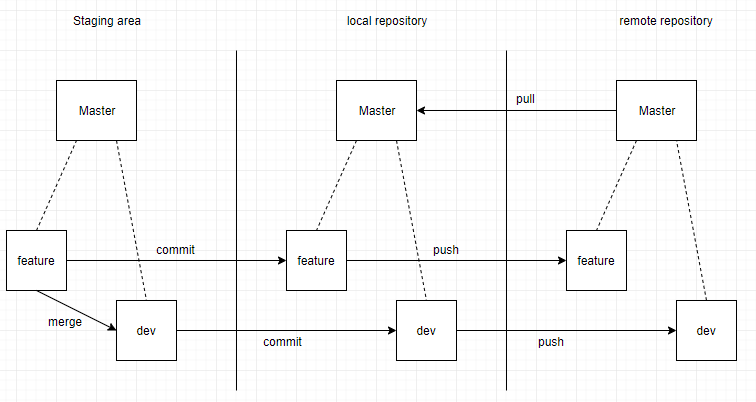
The first three steps in the above figure are to create a folder and enter, **git init** is to initialize a git repository.



**ls -a** is used to view files. As you can see, there is a git hidden directory under MyGit. This directory is used to track file changes. Do not modify this directory, otherwise it will most likely destroy this repository.

# Workspace and Repository

What is a work area? It's where you work, for example, you work in the MyGit directory, and MyGit is a workspace. What about the repository? The .git hidden directory mentioned before, this is a repository, there is a staging area and current branch under the repository, the staging area and branch will be mentioned later.



# Submit a version

When you finish your work in the workspace, you need to submit it. Suppose you create a file called README.txt, and now that you have created it, how do you submit it?

First you have to open **Git Bash** in your workspace and type it in two steps:

$ git add README.txt

$ git commit -m 'add README.txt'

The first step is to add the README.txt file of the workspace to the staging area of the repository. The second step is to submit the contents of the staging area to the local repository. The content after **-m** is the description of this commit. You can also "add" multiple files and submit them one time. When you need to add all the files in the workspace, you can use **git add .** to simplify the operation.

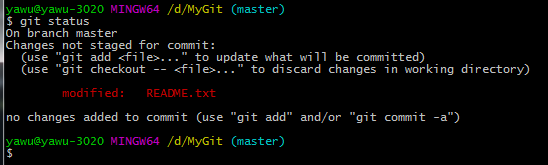
# Version control

## Git status

**git status** lets us know the current state of git, now add the following under README.txt:

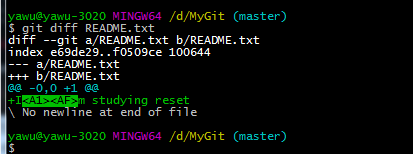
I’m studying reset

Use **git status** to see the status of git:



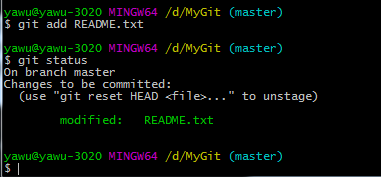
The above prompts that README.txt has been modified, but has not been submitted, you can use **git add** to upload to the staging area, you can also use **git checkout -- <file>** to abandon this modification.

The above picture only says that README.txt has been modified, but it does not explain the modification of the content, then we can use the **git diff** command.



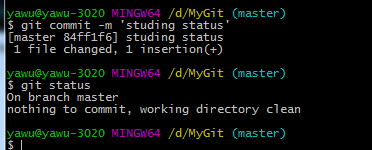
As you can see from the above figure, a line of content has been added.

Submit the changes to the staging area. Then use **git status** to view the status:



You can see the prompt, the change is ready to be submitted, the modified line turns green, indicating that README.txt has been added to the staging area.

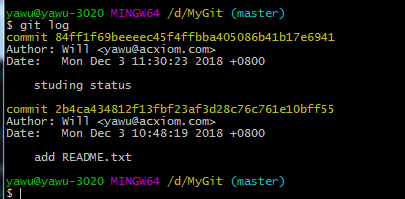
Then submit to the local repository and view the status:



After submitting, you can see that the workspace is clean and there are no files to submit.

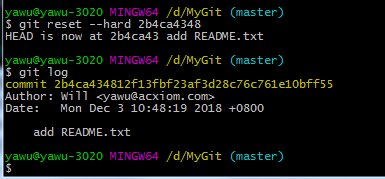
## Version fallback

How many versions have we had so far? Can be viewed with the **git log** command.

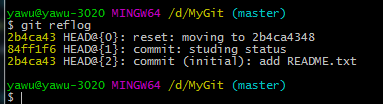


The commit is followed by the version number of each commit, the Author is the username and mailbox you set before, the Date is the commit time, and the last line is the commit description of the commit, which is the content after **git commit -m**.

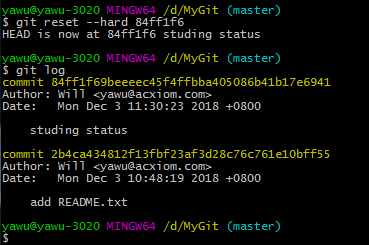
Now I want to go back to the previous version, which is when adding README.txt. You can use **git reset --hard "version number"**, the version number can only be the first few digits, as long as the first few bits can already uniquely locate a version. For example, I can go back to the previous version by typing **git reset --hard 2b4ca434** :



Now look through the **git log** and find that there is only one version left, indicating that it has been rolled back. You can also use the **git reflog** command to view your version record:



Submit twice, once back, and also record the version number of the operation at the time.

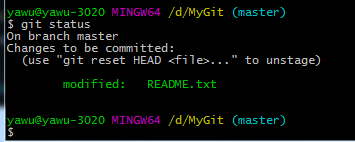


## Discard changes

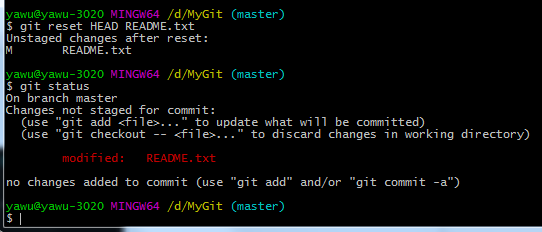
Now add a paragraph to README.txt:

"sun is black"

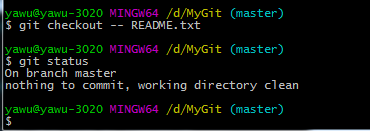
Ok, then you submit it to the staging area, then you find some errors and want to revoke? Git has given us a hint to use **git status** to view:



Git prompts us to revoke with **git reset HEAD <file>**



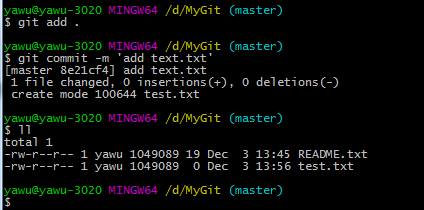
As you can see, there is no content in the staging area. Only the workspace still has changes. You can also use **git checkout -- <file>** to discard the modification of the workspace and make it consistent with the contents of the local repository.



If you have already submitted to the local repository? You can use **git reset –hard “version number”** to go back to the previous version. The above operation you can use **git status** to abtain prompt!

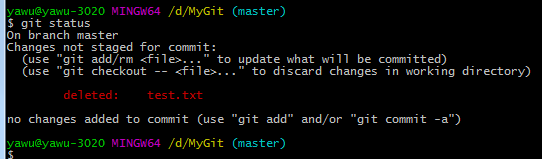
## Delete files

We first create a file test.txt and submit it for later deletion:

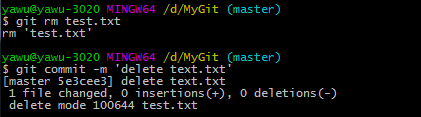


**ll** is used to display the files in the current directory, you can see that there is already test.txt

Now I delete test.txt



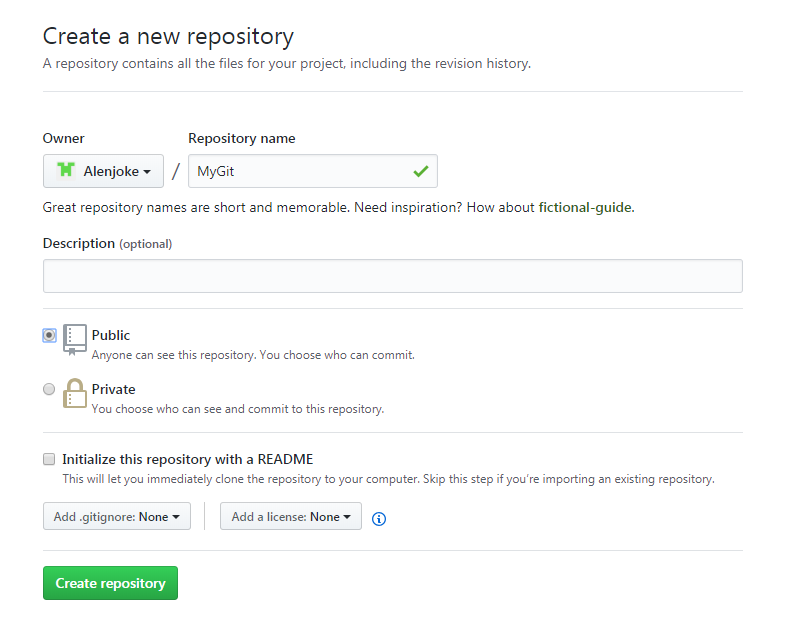
As you can see, after the workspace deletes the file, it needs to be submitted to the staging area with **git rm <file>**, and finally submitted to the local repository.



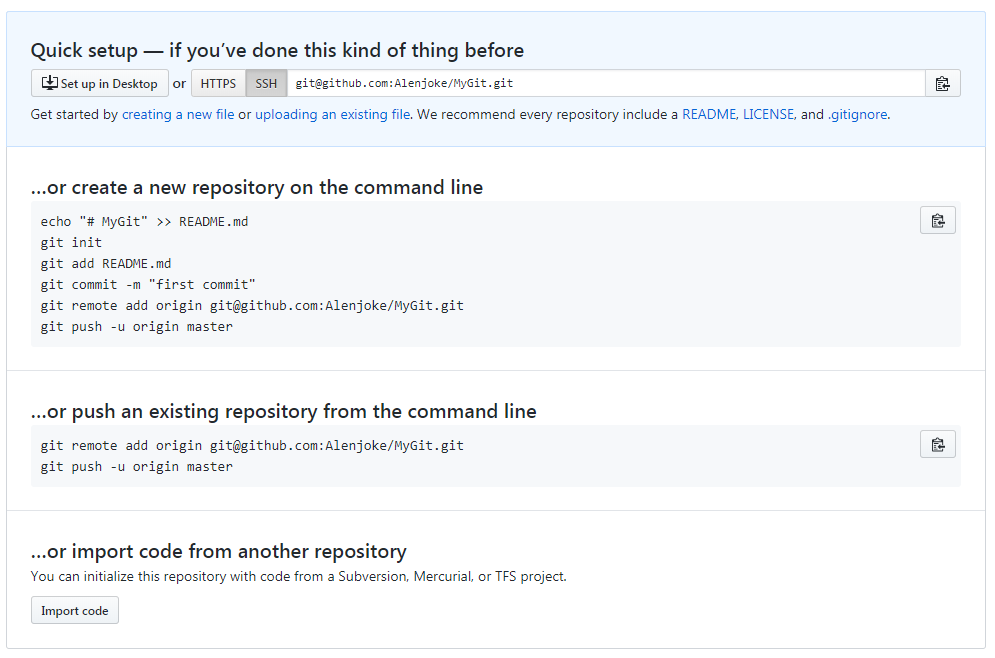
# Remote Repository

Git is a distributed version control system. Naturally, it can't be operated on only one machine. We set up a server, everyone can download code from this machine, and can also commit changes to the remote server. Others can see the modifications you made. Here we use the GitHub website as an example of a remote server.

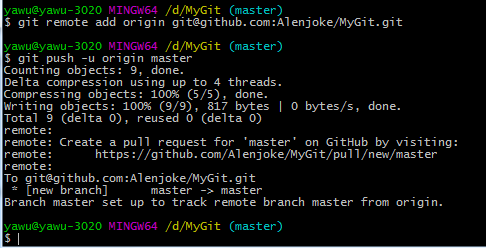
## Create a remote repository



Click **Create repository**



It prompts us to create a repository ourselves, or push one locally, here we choose to push locally.



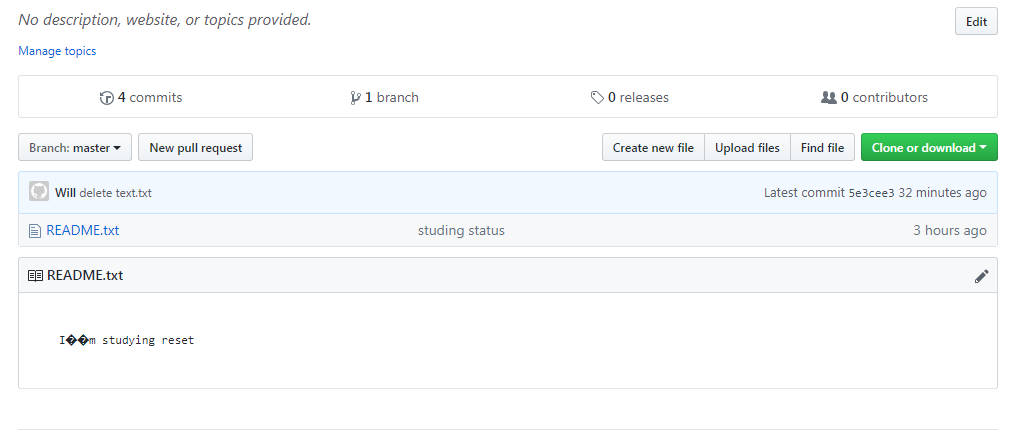
We push the local repository to the remote server.

$ **git remote add origin git@github.com:Alenjoke/MyGit.git**

Associate a local repository with a remote repository

$ **git push -u origin master**

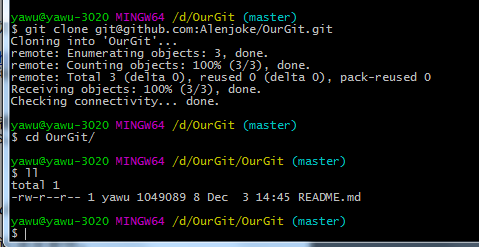
Push the local master to the remote master. Since the remote repository is empty, the first push needs to add the -u command, which is not needed in the future.



You can see that it has been successfully pushed to the remote server, and there are 4 commit records.

## Git clone

If the local project is missing, you can use the **git clone** command to get all the versions on the remote server. For example: **git clone git@github.com:Alenjoke/OurGit.git**



# Branch

## Create a branch from master(default)

What is the use of the branch, I don't know if you noticed it, there is a blue (master) in git bash, which means that we are working on the master branch now, this is a default branch. Suppose we only have this branch. When I work with multiple people, I developed a function, but only half completed. If I upload it to the remote, the code will not run when someone pulls the code, and if I don't submit it, I may face the risk of code loss. At this point the branch works, you can submit the code to your own branch.

Create a branch:

**git branch dev**

Switch branches:

**git checkout dev**

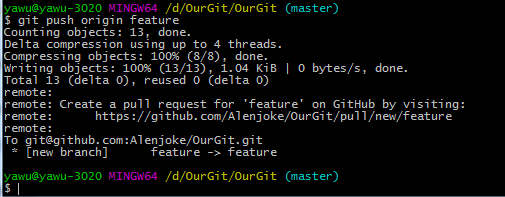
View the branch:

**git branch**



## Push local branch to remote

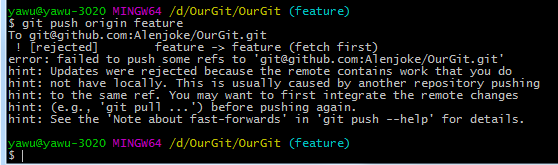
**git push origin “local branch”**



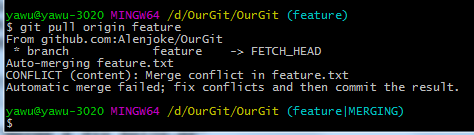
Here I push the feature branch directly to the remote, so the remote will automatically create a feature branch corresponding to my local.

## Branch conflict

Now other people have feature branch locally, and made some changes and submitted them to the remote. At this point, your work on the feature branch is completed, and then submitted to the remote, there will be conflicts.

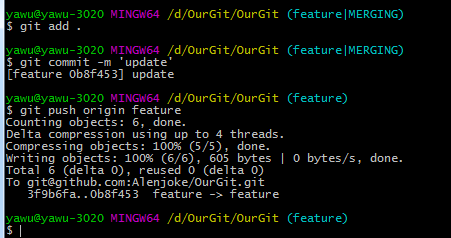


git prompts us to use **git pull** to resolve conflicts.



There is a conflict between git pull and local, we resolve the conflict locally and then submit it.

git pull = git fetch + git merge



Once the conflict is resolved, it can be successfully submitted.