GIT

由来

Linux内核代码需要版本管理工具管理代码，2002年开始，使用的是BitMover公司的BitKeeper这个商用软件。但是Linux社区崇尚的是自由软件相悖。

2005年，Andrew Tridgell对BitKeeper的协议进行逆向工程，BitKeeper作者决定收回无偿使用授权。磋商无果，Linus又找不到合适的版本管理工具，决定自行开发分布式版本管理工具，一个月后，Linux内核代码被Git接管。

2008年，基于WEB使用Git进行版本控制的软件托管服务的网站GitHub上线。

2016年5月9日，11年后，BitKeeper开源，发布在了GitHub上。

2018年6月4日，微软宣布，通过75亿美元的股票交易收购代码托管平台GitHub。

安装

下载对应操作系统的Git客户端版本 https://git-scm.com/downloads

Linux

从RHEL上安装非常简单

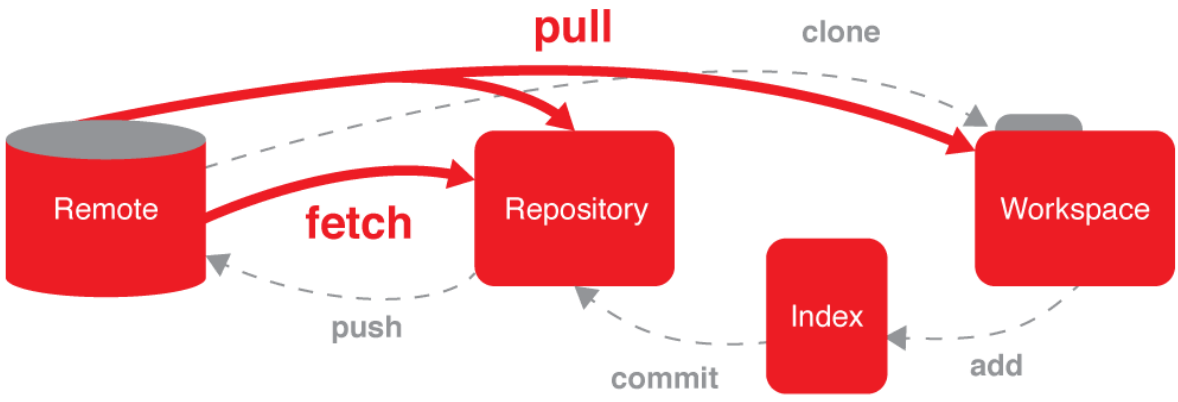
```
# yum install git
```

windows

下载对应的32位或者64版本，点击安装即可

```
$ git --version 查看版本号
```

核心概念



名称	
Repository 仓库、版本库	git初始化后，会在当前目录生成一个.git目录，这就是版本库
Workspace 工作空间、工作区	.git文件所在的目录就是工作区，一般是项目的根目录
index索引	介于工作区和版本库之间，暂存修改的
remote 远程版本库	网络上的另一个版本库，可以和本地库交互

初始化一个版本库

```
$ git init
Initialized empty Git repository in /home/python/magedu/projects/cmdb/.git/
```

在当前目录中增加了一个.git目录，不要自行修改这个目录里面的文件。
当前目录一般是项目的根目录。

添加文件

```
$ echo 'start here' > readme
$ git add readme
```

单个文件添加

这一步是把文件的当前变化增加到暂存区中，也就是以后这个文件需要版本库来跟踪管理，注意这不是提交。
此时，文件还可以继续修改，还可以添加新的被跟踪文件，一定要add才能把这些改变加入到索引中

批量添加

```
$ git add .
```

.点号，代表当前目录，这条命令将**递归**添加当前目录及其子目录所有文件。
只要是目录，就会递归添加该目录下的文件和子目录。

查看状态

```
$ touch .temp

$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#   new file:   readme
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
```

```
#   .temp


$ git status -s
A  readme
?? .temp
```

`-s, --short` 短格式输出

Git的文件分类

- 追踪的Tracked，已经加入版本库的文件
- 未追踪的Untracked，未加入到版本库的未被管理的文件
- 忽略的Ignored，git不再关注的文件，例如一些临时文件

.gitignore文件中，目录以/结尾，行起始的!是取反

.gitignore内容如下：

```
.temp
*.ipynb
.gitignore
__pycache__/
```

忽略文件不需要自己写，Python的已经有了https://github.com/github/gitignore/blob/master/Python.gitignore

`$ wget -O .gitignore https://raw.githubusercontent.com/github/gitignore/master/Python.gitignore`

其它语言的在这里找 https://github.com/github/gitignore

再次看看状态

```
$ git status
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#   new file:   readme
#
```

提交代码

```
$ git commit --help

$ git commit -m 'commit 1'

*** Please tell me who you are.

Run

  git config --global user.email "you@example.com"
```

```
  git config --global user.name "Your Name"

to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: empty ident name (for <python@nodex.(none)>) not allowed
```

如果出现上面的问题，提交失败，就需要先设置一下用户名和邮箱。

这个用户名和邮箱表明这次提交是哪个用户完成的。

设置用户信息

```
$ git config --global user.email "wayne@magedu.com"
$ git config --global user.name "wayne"

$ cat ~/.gitconfig

$ git config --global user.name
$ git config --global user.email
```

```
$ git commit -m "commit 1"
[master (root-commit) 5305651] commit 1
 1 file changed, 1 insertion(+)
 create mode 100644 readme
```

commit 命令：提交更改到版本库

- -m选项：填写本次日志消息，**必须写**。工作中，程序员应该对每一次提交写明做了什么改动

修改后再次提交

```
$ echo "line 2" >> readme

$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   readme
#
no changes added to commit (use "git add" and/or "git commit -a")

$ git commit -m "commit 2"
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   readme
#
no changes added to commit (use "git add" and/or "git commit -a")
```

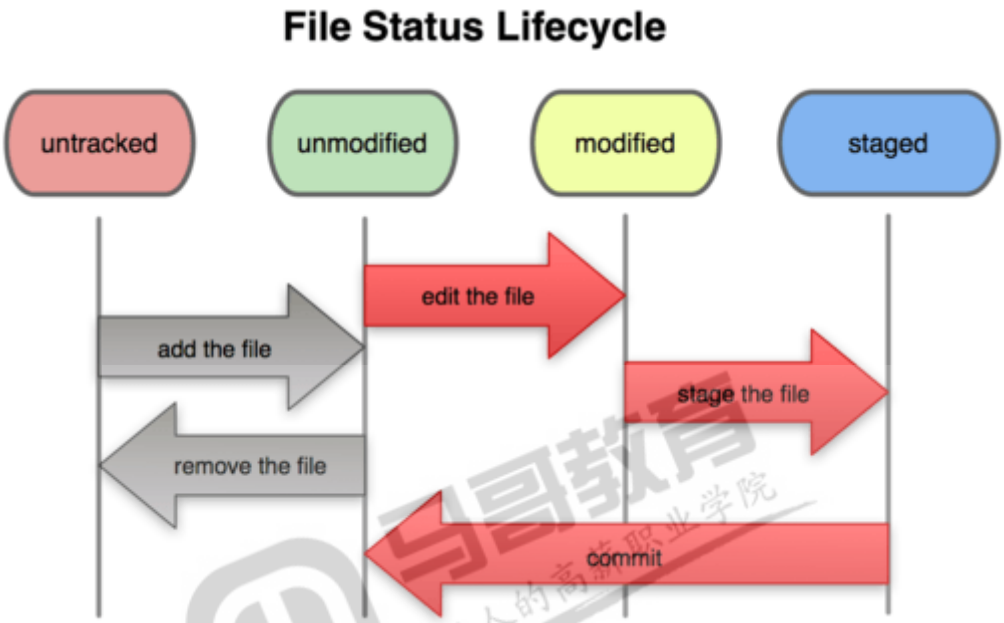
上面commit命令没有任何效果。原因在文件虽然修改但是并没有staged。提示使用git add 或者git commit -a

```
$ git add index.htm

$ git add readme
$ git commit -m "commit 2"
[master e286522] commit 2
 1 file changed, 1 insertion(+)
```

提交成功

文件的生命周期



文件add后，就成为可跟踪文件的未修改状态unmodified，修改后，文件就变成modified状态。再次add后，将变化提交到索引，状态变为staged，这才能提交。提交成功，文件状态从staged变回unmodified。

git的提交

git的提交分为两个步骤：

1. 暂存变更：add作用是把新文件或者文件新的改动添加到一个暂存区stage，也就是加入到index中
2. 提交变更：commit提交的是暂存区中的改动，而不是物理文件目前的改动，提交到当前分支，默认是master分支

也可以使用下面命令，将两步合成一步

```
$ git commit readme
```

如果改动了一批文件，一个个写名字很麻烦，使用下面的命令

```
$ git commit -a
```

-a，--all 会把所有**跟踪的文件**的改动自动暂存，然后commit。上面命令未提交message，会出现一个类似vi命令的操作界面，需要编写message之后，才行。
也可以使用下面的命令，把message信息一并填写了。

```
$ git commit -a -m "message"
```

增补

第二次提交后，才发现忘记了些内容，补充后，不想多次提交

```
$ echo 2.1 >> readme
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   readme
#
no changes added to commit (use "git add" and/or "git commit -a")

$ git add readme

$ git commit --amend -m "commit 2.1"
[master 991c7e5] commit 2.1
 1 file changed, 2 insertions(+)
```

`--amend` 修改，通过创建一个新的commit来replace当前分支的顶部。
也可以在命令中继续使用-m选项直接提交message。

`git log` 查看一下版本库里面提交的历史记录

恢复

如果错误使用了输出重定向，把内容覆盖了，可以使用下面命令恢复

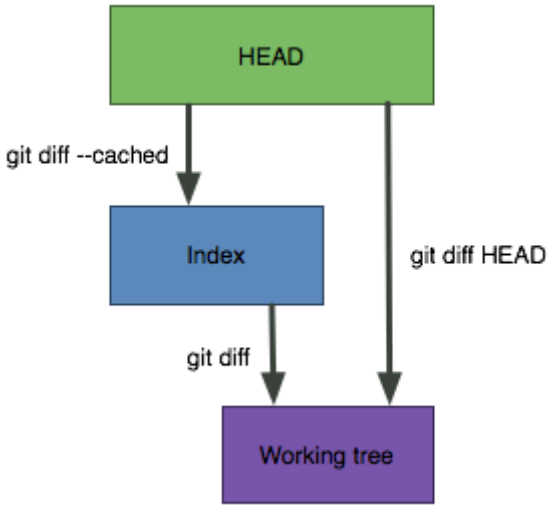
```
$ echo 2.1 > readme
$ git checkout readme
```

从暂存区恢复文件到工作区。

diff比较

查看各种差异

`git diff` 查看被跟踪文件未暂存的修改，比较暂存区和工作区
`git diff --cached` 查看被跟踪文件暂存的修改，比较暂存区和上一次commit的差异
`git diff HEAD`，查看被跟踪文件，比较工作区和上一次commit的差异。HEAD指代最后一次commit



修改文件后，工作区和暂存区、本地仓库都不一样

```
$ echo "line 3" >> readme
$ git status -s
 M readme

$ git diff
diff --git a/readme b/readme
index cb7c1ba..61eb439 100644
--- a/readme
+++ b/readme
@@ -1,3 +1,4 @@
 start here
 line 2
 2.1
+line 3
$ git diff HEAD
diff --git a/readme b/readme
index cb7c1ba..61eb439 100644
--- a/readme
+++ b/readme
@@ -1,3 +1,4 @@
 start here
 line 2
 2.1
+line 3
$ git diff --cached
```

git add后，工作区和暂存区一致了，但它们和本地仓库不一致。工作区 => 暂存区

```
$ git add readme
$ git diff
$ git diff HEAD
diff --git a/readme b/readme
index cb7c1ba..61eb439 100644
--- a/readme
+++ b/readme
@@ -1,3 +1,4 @@
 start here
 line 2
 2.1
+line 3
```

```
$ git diff --cached
diff --git a/readme b/readme
index cb7c1ba..61eb439 100644
--- a/readme
+++ b/readme
@@ -1,3 +1,4 @@
 start here
 line 2
 2.1
+line 3
```

```
git commit后，工作区、暂存区、本地仓库无差别。暂存区 => 本地仓库
$ git commit -m "commit 3"
[master c445809] commit 3
 1 file changed, 1 insertion(+)
$ git diff
$ git diff HEAD
$ git diff --cached
```

HEAD

HEAD可以看做是一个游标，一般是指向当前分支最后一次提交。
HEAD的值存储在.git/HEAD中。

表示	说明
HEAD	指代最后一次commit
HEAD^	指代上一次提交
HEAD^^	指代上上一次提交
HEAD~n	表示倒数第n次提交

检出

checkout 用于（创建）切换分支，或恢复工作区文件。

注意，checkout会重写工作区，这个命令还是有危险性的。

命令	说明
git checkout	列出暂存区可以被检出的文件
git checkout file	从暂存区检出文件到工作区，就是覆盖工作区文件，可指定检出的文件。 但是不清除stage
git checkout commit file	检出某个commit的指定文件到暂存区和工作区
git checkout .	检出暂存区的所有文件到工作区

```
不小心写错了，工作区内容覆盖了
$ echo "random string" > readme
$ git diff
diff --git a/readme b/readme
index 61eb439..6c588a5 100644
--- a/readme
+++ b/readme
@@ -1,4 +1 @@
-start here
-line 2
-2.1
-line 3
+random string

开始恢复工作区，但暂存区不动
$ git checkout readme
$ git diff
```

```
使用当前分支的最后一次commit检出覆盖暂存区和工作区
$ git checkout HEAD readme
```

重置

命令	说明
git reset	列出将被reset的文件
git reset file	缺省行为。重置文件的暂存区，和上一次commit一致，工作区不影响
git reset --hard	重置暂存区与工作区，与上一次commit保持一致

命令	说明
git reflog	显示commit的信息，只要HEAD发生变化，就可以在这里看到
git reset --soft commit	重置当前HEAD为指定commit，但保持暂存区和工作区不变
git reset commit	缺省行为，即--mixed。 重置当前分支的HEAD为指定commit，重置暂存区，但工作区不变
git reset --hard commit	重置当前分支的HEAD为指定commit，同时重置暂存区和工作区，与指定commit一致

```
新建一个文件并提交
$ echo "print('hello magedu')" > app.py
[python@nodex cmdb]$ git add app.py
[python@nodex cmdb]$ git commit -a -m "commit 4 add app.py"
[master 008a263] commit 4 add app.py
 1 file changed, 1 insertion(+)
 create mode 100644 app.py
```

```
重置
$ ls
$ git log
commit 008a263e8ce7bd7b445c35cda0e3832604d7cd6d
Author: wayne <wayne@magedu.com>
Date:   Fri Dec 27 00:54:29 2019 +0800

    commit 4 add app.py

commit c4458097860ed59c78accd28fa10b83f373d00a9
Author: wayne <wayne@magedu.com>
Date:   Fri Dec 27 00:13:44 2019 +0800

    commit 3

commit 991c7e5dc563a72f672af21c50039545bb2fb052
Author: wayne <wayne@magedu.com>
Date:   Thu Dec 26 23:57:38 2019 +0800

    commit 2.1

commit d314f46c08234c2167cdcf35e699701fae248fcd
Author: wayne <wayne@magedu.com>
Date:   Thu Dec 26 23:56:14 2019 +0800

    commit 1
$ git reset --hard d314f46
HEAD is now at d314f46 commit 1
$ git log
commit d314f46c08234c2167cdcf35e699701fae248fcd
Author: wayne <wayne@magedu.com>
Date:   Thu Dec 26 23:56:14 2019 +0800

    commit 1
$ git reflog
d314f46 HEAD@{0}: reset: moving to d314f46
008a263 HEAD@{1}: commit: commit 4 add app.py
c445809 HEAD@{2}: commit: commit 3
991c7e5 HEAD@{3}: commit (amend): commit 2.1
0851d5f HEAD@{4}: commit: commit 2
d314f46 HEAD@{5}: commit (initial): commit 1
$ ls


再次重置
$ git reset --hard 008a263
HEAD is now at 008a263 commit 4 add app.py
$ git log
$ git reflog
$ ls
$ cat app.py
```

reset操作，要慎重。

`git reset --hard [commit]`用于版本回滚、再恢复，但也要慎重使用。

撤销

git revert。

```
修改并提交，发现错了，想要倒回去一个版本
$ echo "line 4" > readme
[python@nodex cmdb]$ git commit -a -m "commit 5, add line 4"
[master c19e863] commit 5, add line 4
 1 file changed, 1 insertion(+), 4 deletions(-)
[python@nodex cmdb]$ cat readme
line 4

撤销最后一次提交，生成新的提交
[python@nodex cmdb]$ git revert HEAD
[master 1bd161b] Revert "commit 5, add line 4"
 1 file changed, 4 insertions(+), 1 deletion(-)
[python@nodex cmdb]$ cat readme
start here
line 2
2.1
line 3
[python@nodex cmdb]$ git log
```

revert主要用在主分支上，保留信息。在工作分支上，可以采用reset。

移动和删除

`git mv src dest` 改名，直接把改名的改动放入暂存区
`git rm file` 会同时在版本库和工作目录中删除文件，真删除
`git rm --cached file` 将文件从暂存转成未暂存，从版本库中删除，但不删除工作目录的该文件，即文件恢复成不追踪的状态
以上都算是改动，必须commit才算真改动了

```
新增一个测试文件
$ echo test > test.py
[python@nodex cmdb]$ git add test.py
[python@nodex cmdb]$ git commit -m "mv and delete"
[master 6f99e21] mv and delete
 1 file changed, 1 insertion(+)
 create mode 100644 test.py
```

改名

```
$ git mv test.py t.py
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   renamed:    test.py -> t.py
#
$ git commit -m "test to t"
[master 10ba387] test to t
 1 file changed, 0 insertions(+), 0 deletions(-)
 rename test.py => t.py (100%)
```

删除文件

```
$ git rm t.py
rm 't.py'
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   deleted:    t.py
#
$ git commit -m "del t.py"
[master dcf5aed] del t.py
 1 file changed, 1 deletion(-)
 delete mode 100644 t.py

$ ls
app.py  readme
```


push到服务器

自己搭建了一个github私服，模拟GitHub
http://yourip:3000/my/test.git

关联远程版本库
`git remote` 列出所有远程仓库
`git remote -v` 详细列出所有远程仓库
`git remote add [shortname] [url]` 指定一个名称指向远程仓库

增加一条名称和仓库url映射

```
$ git remote add origin http://my@192.168.142.140:3000/my/test.git
$ cat .git/config
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
[remote "origin"]
    url = http://my@192.168.142.140:3000/my/test.git
    fetch = +refs/heads/*:refs/remotes/origin/*
```

远程版本库名origin，这是个习惯用法，将建立origin和后面url的映射，这些信息保存在.git/config文件的新的段 [remote "origin"] 中。

注意：`http://my@192.168.142.140:3000/my/test.git`加上用户名，否则push会报401

- `git config --system` 在 `/etc/gitconfig` 文件中读写配置
- `git config --global` 在 `~/.gitconfig` 文件中读写配置
- .git/config 这个文件是 版本库级别 设置文件，这里的设置具有最高优先级

推送数据

```
$ git push -u origin master
Password:
Counting objects: 7, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (7/7), 583 bytes, done.
Total 7 (delta 0), reused 0 (delta 0)
To http://my@192.168.142.140:3000/my/test.git
 * [new branch]      master -> master
Branch master set up to track remote branch master from origin.
```

输入密码就可以连接到远程仓库了。
私有的仓库，必须登录，只能用户自己看，为了方便，修改为公有的。

```
$ git push origin master    # 指定推送到的远程主机和分支
$ git push origin           # 指定当前分支推送到的主机和对应分支
$ git push -u origin master # 指定远程默认主机和分支
$ git push                  # simple方式，默认只推送当前分支到默认关联的远程仓库
```

-u 第一次远程推送的时候加上，以后就可以不使用-u参数，可以git push origin master，也可以git push都使用默认。

```
$ echo "line 5" >> readme

$ $ git commit -a -m "commit 5, append line 5"
[master eeff550] commit 5, append 4
 1 file changed, 1 insertion(+)

$ git push origin master
或者
$ git push
```

从远程库克隆

这一次使用git协议连接远程库。

为了演示跨平台，这里使用windows系统。

建议使用Git的windows客户端的 `git bash` ，它含有常用ssh命令

配置本地用户名、邮箱

```
$ git config --global user.name "wayne"
$ git config --global user.email "wayne@magedu.com"
$ cat ~/.gitconfig
[user]
    name = wayne
    email = wayne@magedu.com
```

删除windows当前用户.ssh文件夹

`$ ssh-keygen -t rsa -C "wayne@magedu.com"`

-t 加密算法类型

-C comment 描述信息

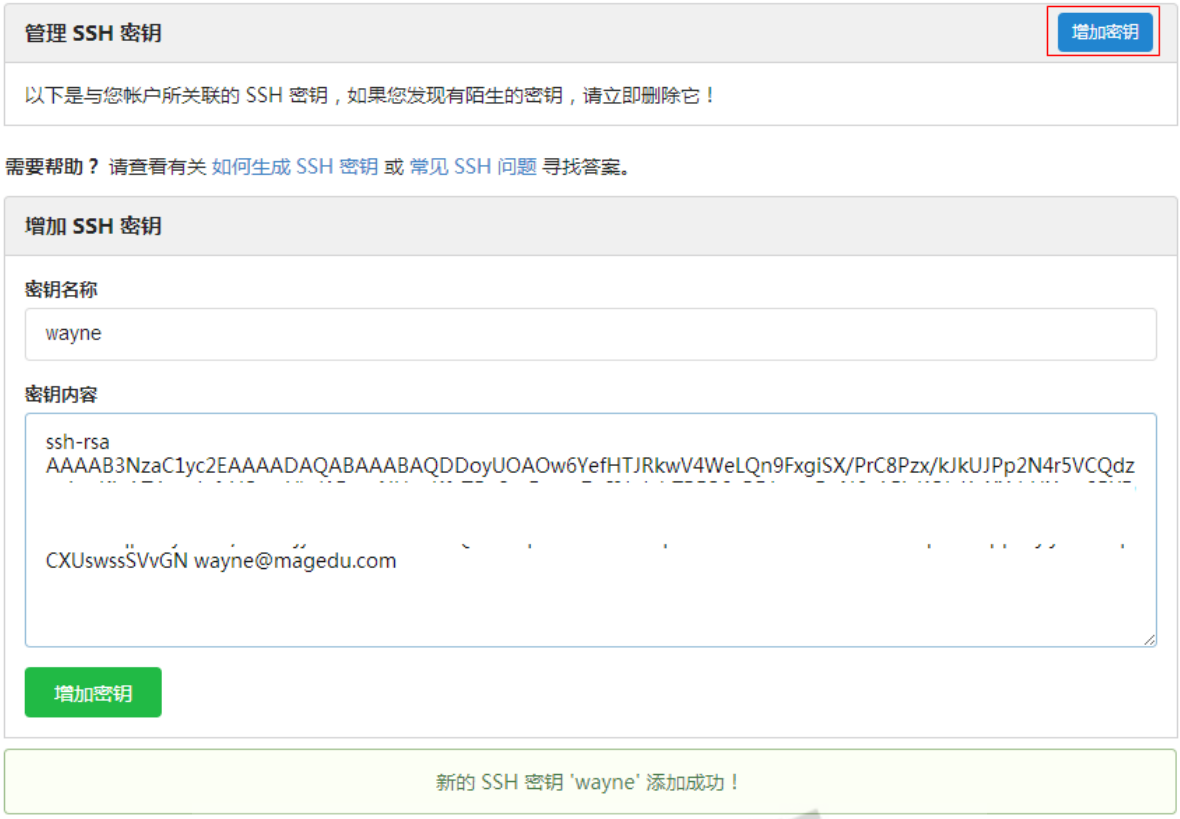
```
$ ssh-keygen -t rsa -C "wayne@magedu.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Administrator/.ssh/id_rsa): # 直接回车
Enter passphrase (empty for no passphrase): # 直接回车
Enter same passphrase again: # 直接回车
Your identification has been saved in /c/Users/Administrator/.ssh/id_rsa. # 私钥
Your public key has been saved in /c/Users/Administrator/.ssh/id_rsa.pub. # 公钥
The key fingerprint is:
SHA256:ZxALWxgiq1UUw6TpS+p/hBeTeYOmAbIRVQNUZZ0fUxs wayne@magedu.com
The key's randomart image is:
+---[RSA 2048]----+
|.+=B@o=+o. .E    |
|o. *.=.+ooo  o   |
|.o*   = o. o.    |
|.+ . B o ..      |
|. o = + S o      |
| o + o   o       |
|. . o            |
|.    .           |
| ....            |
+----[SHA256]-----+

$ cd
$ ls .ssh
id_rsa  id_rsa.pub

$ pwd
/c/Users/Administrator
```

打开gogs的用户设置 -> SSH密钥

打开公钥文件~/.ssh/id_rsa.pub，将内容贴入“密钥内容”框中，点击“增加密钥”



那么SSH登录的用户使用的链接如下图

SSH连接远程库

在windows上找一个空目录，执行下面的克隆命令。

注意，第一次使用ssh连接有提示，敲入yes。

```
$ git clone git@192.168.142.140:my/test.git
Cloning into 'test'...
remote: Counting objects: 28, done.
remote: Compressing objects: 100% (20/20), done.
remote: Total 28 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (28/28), done.
```

克隆成功。
下面就可以使用这个初始的项目文件开发了。

注：Linux和windows下交替演示，是想说明，git客户端无所谓在什么操作系统。