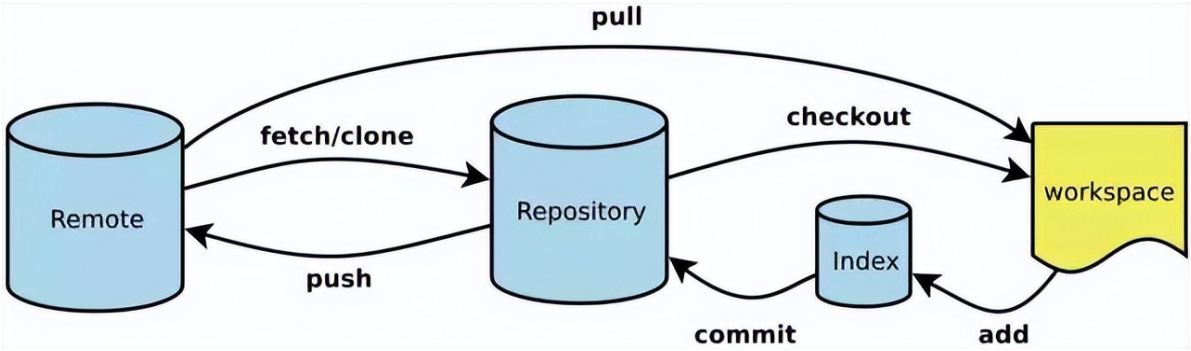
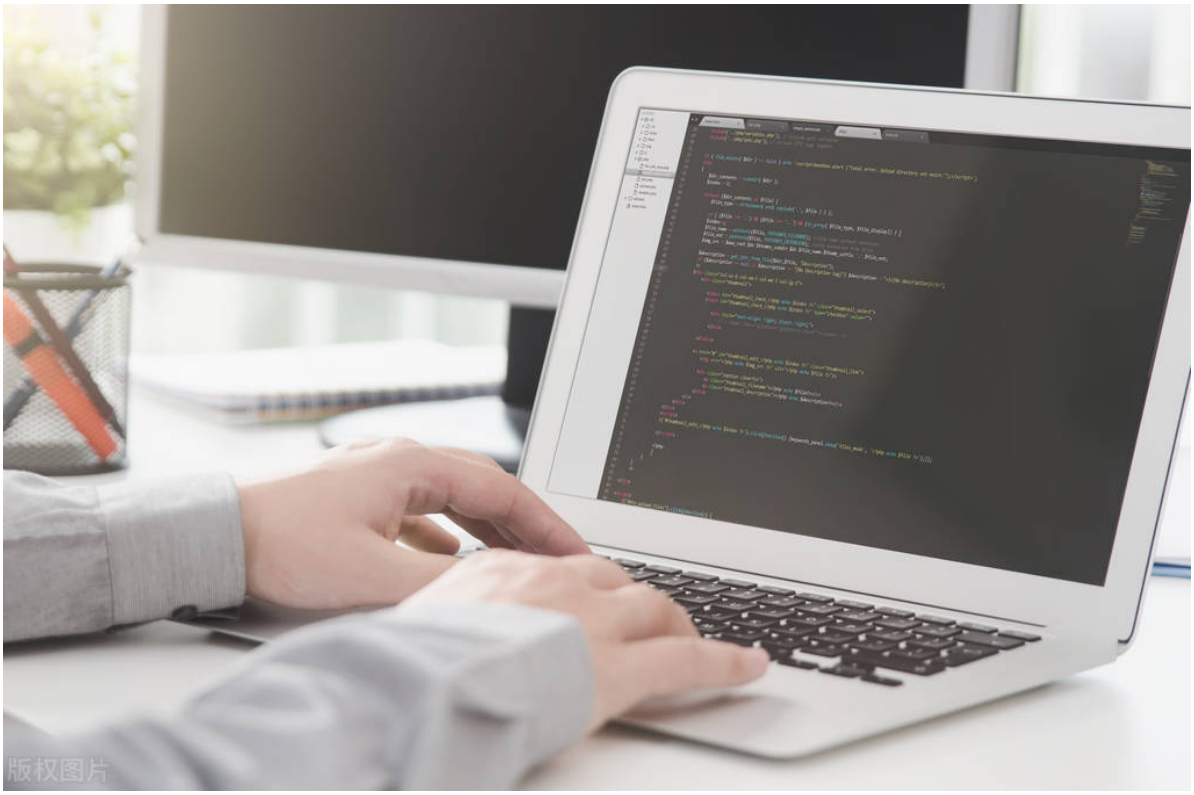
git工作原理

git是本地进行版本管理，github是将本地的程序推送到Github网站存储。



git有四个区，工作区Workspace就是我们平时写代码的目录；Index / Stage暂存区可以理解为当我们完成一个阶段的任务后存储一个快照，但是并没有生成版本；Repository仓库区是真正创建一个历史版本，以后有需要可以随时回退；Remote远程仓库可以自己搭建也可以使用Github网站，将最终的版本推送上去，**可以开源与所有程序员共享，也可以是私有仓库，仅供团队内部协同开发。**



创建仓库

首先新建一个目录，cd切换到目录，然后执行命令：git init 初始化：

```
hioier@pc:~$ mkdir mygit
hioier@pc:~$ cd mygit/
hioier@pc:~/mygit$ git init
Initialized empty Git repository in /home/hioier/mygit/.git/
hioier@pc:~/mygit$ ls -a
.  ..  .git
```

当目录中出现隐藏目录.git时，表示创建git仓库成功。**.git文件夹存储当前项目的历史版本和管理数据，每个git用户都存储完整的历史版本，所以git是分布式的。**

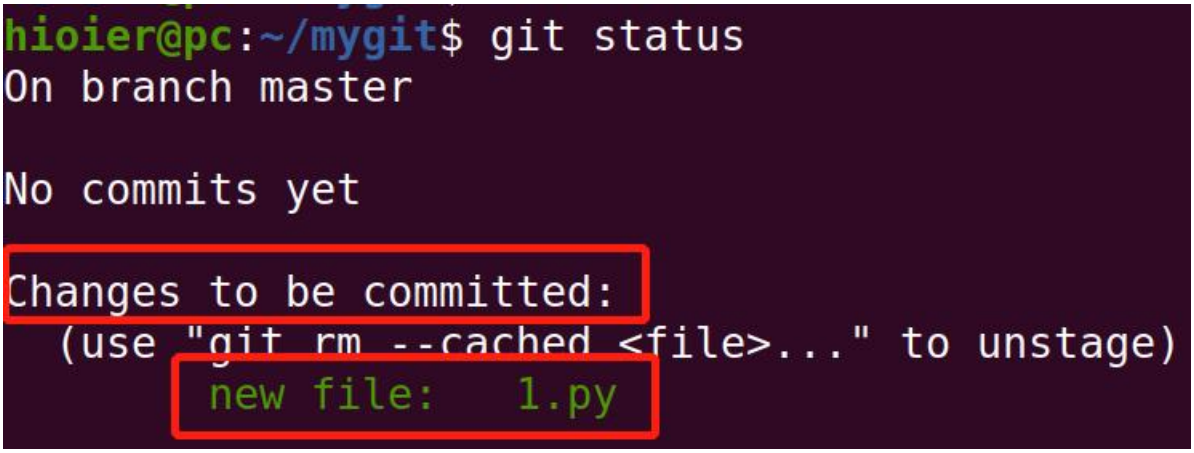
```
.git/
├── branches
├── COMMIT_EDITMSG
├── config
├── description
├── HEAD
├── hooks
│   ├── applypatch-msg.sample
│   ├── commit-msg.sample
│   ├── fsmonitor-watchman.sample
│   ├── post-update.sample
│   ├── pre-applypatch.sample
│   ├── pre-commit.sample
│   ├── pre-merge-commit.sample
│   ├── prepare-commit-msg.sample
│   ├── pre-push.sample
│   ├── pre-rebase.sample
│   ├── pre-receive.sample
│   └── update.sample
├── index
├── info
│   └── exclude
├── logs
│   ├── HEAD
│   └── refs
│       └── heads
│           └── master
├── objects
│   ├── 13
│   │   └── 3d5f1be74ef4133406c743f8f75b41994c00b8
│   ├── 14
│   │   └── 4d8a3e9ffe2be45209c41e1057e0d726b3abcf
│   ├── 16
│   │   └── e0e53dc46f3b758d92b34e5770b7bc1af3583e
│   ├── 30
│   │   └── fb3d089b2a14f02c25dedc5dbf7b37004db7d7
│   ├── 33
│   │   └── 0bc4267fbb6436e54ae8885dbb5322a23d7a90
│   ├── 70
│   │   └── 6d79f4049a0cf035564f09a73724535a143d90
│   ├── 76
│   │   └── 342f7f02603601ffb0c263aa0ccbbea3da1dea
│   ├── 82
│   │   └── 5a804c47b1f4cd529f5282dc7577e48ae4718c
│   ├── ff
│   │   └── 5a6e3adb68a0307c0e1c248f47fd506be00736
│   ├── info
│   └── pack
```

.git也叫做附属于该仓库的工作树。

添加至暂存区

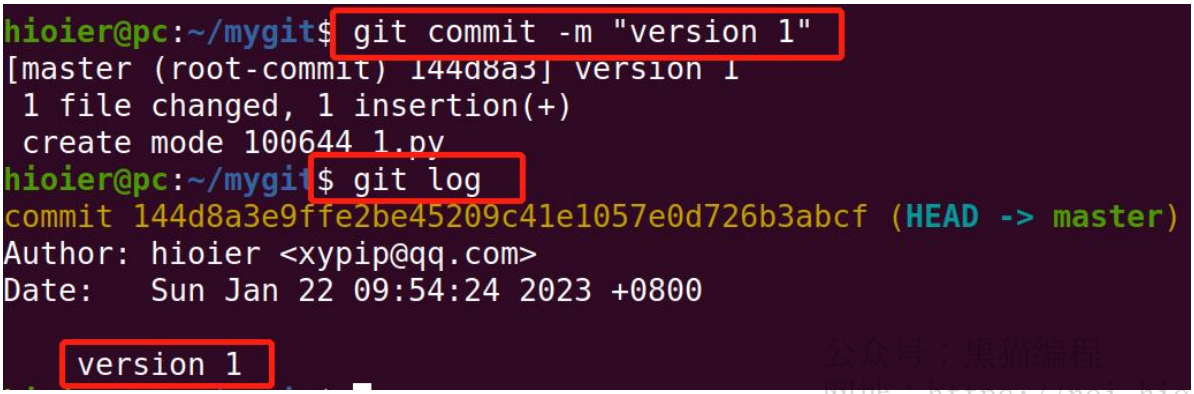
创建一个文件 1.py，然后写入内容：

执行git add 文件名将指定文件添加至暂存区，git status显示所有需要提交的文件，1.py是新建的文件，写入一行代码，就会显示和之前的提交内容有差异（第一次之前没有任何提交）。



commit生成历史版本

git commit -m 备注，在Repository仓库区创建历史版本，git log查看历史版本：



继续在文件中添加一行“version2”，再次查看就会出现两个版本：

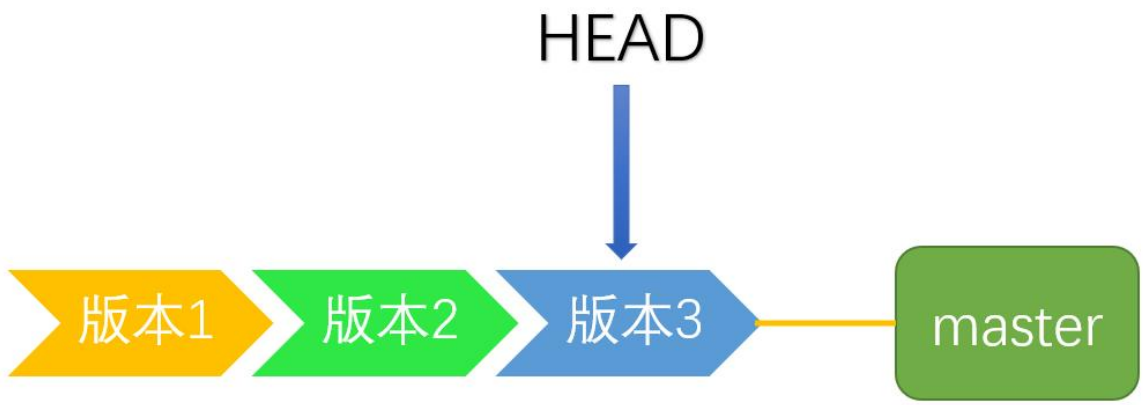
```
hioier@pc:~/mygit$ git add 1.py
hioier@pc:~/mygit$ git commit -m "version 2"
[master 16e0e53] version 2
 1 file changed, 1 insertion(+)
hioier@pc:~/mygit$ git log
commit 16e0e53dc46f3b758d92b34e5770b7bc1af3583e (HEAD -> master)
Author: hioier <xypip@qq.com>
Date:   Sun Jan 22 09:58:01 2023 +0800

    version 2

commit 144d8a3e9ffe2be45209c41e1057e0d726b3abcf
Author: hioier <xypip@qq.com>
Date:   Sun Jan 22 09:54:24 2023 +0800

    version 1
```

下面重复操作，生成3个版本，大家自己练习一下。



当前，都处于master主分支，HEAD就是一个来回移动的指针。

reset版本回退

当前我有三个版本，要回退到之前版本：

```
hioier@pc:~/mygit$ git log
commit 133d5f1be74ef4133406c743f8f75b41994c00b8 (HEAD -> master)
Author: hioier <xypip@qq.com>
Date:   Sun Jan 22 09:59:55 2023 +0800

    version 3

commit 16e0e53dc46f3b758d92b34e5770b7bc1af3583e
Author: hioier <xypip@qq.com>
Date:   Sun Jan 22 09:58:01 2023 +0800

    version 2

commit 144d8a3e9ffe2be45209c41e1057e0d726b3abcf
Author: hioier <xypip@qq.com>
Date:   Sun Jan 22 09:54:24 2023 +0800

    version 1
```

第1种 方案：**git reset --hard 版本号**（版本号取前几位就可以，一般选择前8位）回退到版本2：

```
hioier@pc:~/mygit$ git reset --hard 16e0e5
HEAD is now at 16e0e53 version 2
hioier@pc:~/mygit$ cat 1.py
print("version1")
print("version2")
```

git log发现，版本3的信息没有了，要用git reflog查看操作记录：

```
hioier@pc:~/mygit$ git reflog
16e0e53 (HEAD -> master) HEAD@{0}: reset: moving to 16e0e5
133d5f1 HEAD@{1}: commit: version 3
16e0e53 (HEAD -> master) HEAD@{2}: commit: version 2
144d8a3 HEAD@{3}: commit (initial): version 1
```

然后，相同方案再回到版本3。

```
hioier@pc:~/mygit$ git reset --hard 133d5f1
HEAD is now at 133d5f1 version 3
```

第2种 方案：git reset --hard HEAD^（^的个数代表回退到之前哪个版本，只能向之前的版本回退，比如现在在版本3，回退到版本1就是两个^）

```
hioier@pc:~/mygit$ git reset --hard 133d5f1
HEAD is now at 133d5f1 version 3
hioier@pc:~/mygit$ git reset --hard HEAD^^
HEAD is now at 144d8a3 version 1
```

第3种 方案：git reset --hard HEAD~n（回到到前n个版本）

```
hioier@pc:~/mygit$ git reset --hard 133d5f1
HEAD is now at 133d5f1 version 3
hioier@pc:~/mygit$ git reset --hard HEAD~2
HEAD is now at 144d8a3 version 1
```

简洁显示和显示差别

git log信息简洁显示：

```
hioier@pc:~/mygit$ git log --pretty=oneline
133d5f1be74ef4133406c743f8f75b41994c00b8 (HEAD -> master) version 3
16e0e53dc46f3b758d92b34e5770b7bc1af3583e version 2
144d8a3e9ffe2be45209c41e1057e0d726b3abcf version 1
```

对比HEAD^（即上一个版本）和工作区中当前文件版本不同：

```
hioier@pc:~/mygit$ git diff HEAD -- 1.py
hioier@pc:~/mygit$ git diff HEAD^ -- 1.py
diff --git a/1.py b/1.py
index 706d79f..30fb3d0 100644
--- a/1.py
+++ b/1.py
@@ -1,2 +1,3 @@
 print("version1")
 print("version2")
+print("version3")
```

对比HEAD当前版本和HEAD^^前两个版本之间的不同：

```
hioier@pc:~/mygit$ git diff HEAD HEAD^^ -- 1.py
diff --git a/1.py b/1.py
index 30fb3d0..76342f7 100644
--- a/1.py
+++ b/1.py
@@ -1,3 +1 @@
 print("version1")
-print("version2")
-print("version3")
```

撤销修改

git管理的文件的修改，它只会提交暂存区的修改来创建版本。比如，我们在工作区增加version4，但是没有add到暂存区，直接commit，发现并没有更改的内容可以提交：

```
hioier@pc:~/mygit$ git status
On branch master
nothing to commit, working tree clean
hioier@pc:~/mygit$ cat 1.py
print("version1")
print("version2")
print("version3")
hioier@pc:~/mygit$ vim 1.py
hioier@pc:~/mygit$ cat 1.py
print("version1")
print("version2")
print("version3")
print("version4")
hioier@pc:~/mygit$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   1.py

no changes added to commit (use "git add" and/or "git commit -a")
hioier@pc:~/mygit$ git commit -m "version 4"
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   1.py

no changes added to commit (use "git add" and/or "git commit -a")
```

然后，git checkout -- 文件名 撤销修改：

```
hioier@pc:~/mygit$ git checkout -- 1.py
hioier@pc:~/mygit$ cat 1.py
print("version1")
print("version2")
print("version3")
```

但是，如果我现在已经add到暂存区，应该如何修改?

```
hioier@pc:~/mygit$ git add 1.py
hioier@pc:~/mygit$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   1.py
```

git reset HEAD 文件名，撤销暂存区修改，再撤销工作区修改：

```
hioier@pc:~/mygit$ git reset HEAD 1.py
Unstaged changes after reset:
M       1.py
hioier@pc:~/mygit$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   1.py

no changes added to commit (use "git add" and/or "git commit -a")
hioier@pc:~/mygit$ git checkout -- 1.py
hioier@pc:~/mygit$ cat 1.py
print("version1")
print("version2")
print("version3")
```

删除文件

现在，我要删除1.py，但是后悔了，撤销工作区修改，再找回来：

```
hioier@pc:~/mygit$ rm 1.py
hioier@pc:~/mygit$ git checkout -- 1.py
hioier@pc:~/mygit$ ls
1.py
```

另一种是下定决心要删除，执行git rm 文件名，如果忘记文件名可以通过git status查看：

```
hioier@pc:~/mygit$ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    1.py

no changes added to commit (use "git add" and/or "git commit -a")
hioier@pc:~/mygit$ git rm 1.py
rm '1.py'
hioier@pc:~/mygit$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    1.py
```

然后不需要执行add，直接commit生成历史版本：

```
hioier@pc:~/mygit$ git commit -m "delete 1.py"
[master 24a52f3] delete 1.py
 1 file changed, 3 deletions(-)
 delete mode 100644 1.py
```