1. git是一个分布式的版本控制工具。由Linux的作者Linus创造的。在git创造之前，比较流行的控制软件为SVN(Subversion)， 相比为SVN, Git更快，Git是分布式的，并且Git在克隆的时候可以克隆整个项目，而SVN不可以。Github便是基于Git来创造的在线网站, 已经有许多的大型项目在Github上托管例如python, bootstrap等。

2. 从<https://git-scm.com下载Git>安装以后，便可以在控制台使用git命令。git –version查看git安装的版本。

3. git 基本配置:

git config --global user.email “xxxx” 设置用户邮箱

git config --global user.name“xxxx” 设置用户名称

P.s 这样设置是全局的设置，如果没有在一个代码库中没有具体的设置，那么将使用全局的配置

4.

git init 文件夹名字 🡪 将会在当前路径下创建一个新的代码库文件夹，也就是建立了一个新的repository

在该文件夹下会有.git文件夹，里面将会有git相关的文件以及文件夹。

.git/branches --- 代码库的分支情况

.git/config --- 代码库的本地配置情况

.git/description ---代码库的描述情况

.git/hooks --- 里面可以存放脚本，用于在push, update，commit操作之前执行的函数

.git/ref ---保存一些引用信息

.git/index –-只保存变化，未来将应用到代码库中

P.s 一些在linux下面，在文件或者文件夹下面以.开头的都是隐藏文件，.git里面的内容一般不需要手动进行更改。隐藏文件在linux下查看使用ls -a 选项。

5.git status 展示代码库的状态变化。

git add [filename] 添加代码库的变化，包括文件内容的更改，文件添加等。这些变化将放到index中。

git rm [filename] 和git add一样性质，将变化放到index中，只是不是修改添加，而是删除文件

git commit -m “需要添加的注释” 提交状态到代码库，默认提交到master分支上

git log 展示git的操作历史

git rm --cached [filename] 将某文件从stage的状态中取出

6.git status 展示代码库的状态，如果一个文件是Untracked file， 那么代表的意思是这个文件状态不明，没有被git追踪。 但是有些时候，有些文件特别是非源码文件，而是源码文件产生的副产品的时候，我们不想这个文件被追踪，

-- 可以在. git/info/exclude中添加需要忽略的文件写入，支持通配符。被忽略的文件不会被status显示，也不会被add更新

-- 也可以在. gitignore文件中，中添加需要忽略的文件写入，支持通配符。被忽略的文件不会被status显示，也不会被add更新。 .gitignore文件不需要自己慢慢写，在github上有模版，直接列出了各种编程语言常见的需要忽略的文件，直接使用便行。

git status 展示所有文件的状态

git status <target\_file> 展示特定文件的状态

状态展示会有三种：

命令中”Changes to be committed“中所列的内容是在git所追踪的文件，并且已经通过add到Index(也就是stagin area，也就是暂缓区)中的内容，等待commit之后进入本地Git Directory。

命令中“Changes not staged for commit”中所列的内容是被git追踪，在Working Directory中的内容发生了改变，等待add之后将进入Index。

命令中“Untracked files”中所列的内容是尚未被Git跟踪的内容，add之后进入Index

8.

在本地环境下，通过git add将变化从working directory里面提交到Staging Area里面，再通过git commit提交到repository里面。 如果内容在working directory变化，但是没有提交到staging area里面，git commit是不会提交的。

9.

Github与Gitlab的区别

Github上可以上传源码，如果免费的则需要进行开源，如果是私密的repository则需要付费。现被微软收购。

Gitlab有两种版本，Community和Enterprise的，是一个开源的产品。

可以免费建立私人的repository.

可以下载Gitlab Community源码，然后在自己服务器上自行搭建Gitlab服务器用于保护源码。

10.

我们在Github上创建repository以后，可以通过

git clone 把Github上的repository的代码库拿到本地上

git push [remote-name] [branch-name] 把本地的对应branch的repository的变更放到remote-name所指定的远端的所指定的branch-name上. 如果没有指定[remote-name][branch-name]那么将会push到origin的master上。

git remote -v 显示远端github 上，fetch和push的指向url, 也就是远端代码库的地址

git remote show origin 显示origin主机上分支的情况

git remote add [远端名] [远端地址] 添加一个新的远端地址，并且命名为远端名

11.

分支的作用是在master分支以外，根据不同的开发目的在不同的分支上开发，最后合并到master分支当中。

git branch 查看当前代码库所在的分支

git branch -a 查看所有分支包括本地以及远程

git branch -D <分支名> 强制删除本地指定分支，即使他还没有被merged

git branch -d <分支名> 删除本地指定分支，被merged才可以顺利删除

git branch <分支名> 建立新的分支

git checkout <分支名> 进入某分支

git checkout -b 创建分支名并且进入某分支

git merge 分支名 将某分支merge到当前分支

P.s 在分支1所造的操作，所commit的内容，在其他分支上不会收到修改的影响。

P.s 如果将一个分支merge到另一个分支的时候，两个分支都对同一个文件修改，则会产生冲突。此时，git会在文件中显示冲突的不同，我们需要手动的编辑改文件来解决冲突，然后进行commit完成merge.

12.

.git/FETCH\_HEAD 保存github远端上最后一次commit的哈希值，通过git fetch origin 分支名更新

.git/HEAD 保存本地最后一次commit的哈希值，通过git commit 来更新

13. Fetch操作

git fetch <remote\_repo\_name> <branch\_name> 返回远端地址对应的分支的最后一次commit的哈希值，存放在. git/FETCH\_HEAD中，我们使用这个值和本地的代码库最新一个commit的哈希值进行对比。如果相同则表示，本地代码库的该分支和远端的相同，不需要从服务器更新或者更新服务器。

e.g

----git fetch origin master 只取回远端master分支的更新

7d402cf42748789be947e81a55c916ed0e2ffe7d branch 'master' of orbw-git.ca.alcatel-lucent.com:xinde/Django

git fetch 不加参数，不指定repo\_name和branch\_name则取回远程主机的所有分支更新到FETCH\_HEAD中

e.g:

-----git fetch

-----cat .git/FETCH\_HEAD

[root@galvatron Django]# cat .git/FETCH\_HEAD

7d402cf42748789be947e81a55c916ed0e2ffe7d branch 'master' of orbw-git.ca.alcatel-lucent.com:xinde/Django

faa74fe419dbbb05ae30f1244c5566c3a58916f5 not-for-merge branch 'HuiBranch' of orbw-git.ca.alcatel-lucent.com:xinde/Django

8e2ca0e469338eb127e899632bb7bfd142b47972 not-for-merge branch 'huihj/NSP-0000-GalvatronBuxFix' of orbw-git.ca.alcatel-lucent.com:xinde/Django

git merge FETCH\_HEAD 将通过Fetch\_HEAD所代表的远端的commit应用到本地的working directory当中。

14.

git fetch是将远程主机的最新内容拉到本地，用户在检查了以后决定是否合并到工作本机分支中。

而git pull 则是将远程主机的最新内容拉下来后直接合并，即：git pull = git fetch + git merge，这样可能会产生冲突，需要手动解决。

15.

Tag的应用，tag主要是对commit进行标记。我们可以在github上对某次提交打上标记。同时在本地

git tag -l 展示所有标签

git fetch -t 获取远端的所有标签

git tag <标签名> 为了最近一次提交打上标签

git tag -d <标签名> 本地删除标签

16.

git命令下-h选项表示列出该命令的帮助信息。

17.

gitignore的创建方法：

首先,.gitignore文件应该是创建在你的项目的根目录下面的，你可以使用vi .gitignore并且添加如下需要忽略的内容：

1 node\_modules

2 dist

3 .tmp

4 bower\_components

5 test

18. 对与已提交的不必要文件的处理方法

有时候当我们明白这个gitignore之后才发现我们已经提交不必要的文件了，不愿意让那些文件存在我们库里

git rm -r --cached <file\_directory> #删除远端的记录，但是不删除本地的文件, 将所包含的文件从repository取出，然后再git push告诉远端，那么远端将删除该文件。

19.

git作为支持分布式版本管理的工具，它管理的库（repository）分为本地库、远程库。

git commit操作的是本地库，git push操作的是远程库。

git commit是将本地修改过的文件提交到本地库中。

git push是将本地库中的最新信息发送给远程库。

20.

Git 在1.8版本以后，将Changed but not updated变为changed not staged for commit

21.

git add . ：他会监控工作区的状态树，使用它会把工作时的所有变化提交到暂存区，包括文件内容修改(modified)以及新文件(new)，但不包括被删除的文件。

git add -u ：他仅监控已经被add的文件（即tracked file），他会将被修改的文件提交到暂存区。add -u 不会提交新文件（untracked file）。（git add --update的缩写）

git add -A ：是上面两个功能的合集（git add --all的缩写）

git add <文件> : 只将该文件提交到index

22.

.git/config 配置文件里面的有url的设置，这个url的设置可以有两种类型。第一种是https, 第二种是ssh类型。并且我们可以进行修改，当进行修改以后如果是https那么每次push的时候便需要输入用户名和密码，如果是ssh那么将会使用我们的SSH key进行连接。

23.

Git的pull request和git pull是不一样的概念。 Pull request是当我们fork别人的代码库以后，在自己的代码库中修改后，向对方请求修改，等待对方接受将修改放入自己库中的请求。而git pull是git的一个命令，用于将修改同步自己的代码库中。

24.

从github上clone 项目是不需要权限限制的，这个因为github为开源平台，查看他人的代码是完全开放的。但是进行修改的时候便要进行fork+pull操作。

25.

Github还可以创建issue, 在这个issue当中，可以添加该issue的信息， 设置该issue的标签lable,以及评论或者在pull request中引用等等。

26.

Github上的wiki可以为该github的项目提供一些信息，wiki创建的内容可以作为该github项目的主页。

27.

Github上有organization，team等权限设置。这些权限设置的作用为，可以在github上建立一个组织然后在组织内建立各种project,然后分配组员，组员可以有不同的权限等。

28.

Github Gist是Github的一个子服务，最简单的功能就是分享代码片段。譬如可以把一些代码片段、待处理任务放到Gist上进行分享和管理。或者发gist的链接给组员查看等一些数据阅读。

29.

Github 默认情况下是公开的， 如果变为私密则需要收费。

GitHub Enterprise 也是收费的， 可以搭建在公司内部。

Gitlab Community 开源免费，可以在公司内部搭建也可以选择在gitlab云上搭建。

Gitlab Enterprise 收费，可以在公司内部搭建也可以选择在gitlab云上搭建。

30.

Git diff 命令, 用于展示前后两个文件的异同。特别的如果缺少某个对比目标, git会有默认值指向。

HEAD表示本地代码库中的最新状态， FETCH\_HEAD表示远端库中最新状态， --cache表示暂存区， 什么都不指定为本地工作区

比较工作区与暂存区的所有文件差异

　　git diff 不加参数即默认比较工作区与暂存区

比较暂存区与最新本地版本库（本地库中最近一次commit的内容）

　　git diff --cached  [<path>...]

比较最新本地版本库和工作区的文件差异

　　git diff HEAD [<path>...]

比较工作区与指定commit-id的差异

　　git diff commit-id  [<path>...]

比较暂存区与指定commit-id的差异

　　git diff --cached [<commit-id>] [<path>...]

比较两个commit-id之间的差异

git diff [<commit-id>] [<commit-id>]

31.

git commit 的时候可以加入留言，比较好的留言格式将会是：

第一行总结，可以作为commit message的标题。

第二段描述问题，以及改变的方法。

第三段写出该commit主要针对哪个带好的fix或者issue id.

32.

Git commit提交操作比较好的操作习惯时，分次一定量的提交。这有利于理解commit的内容而且比较好追踪回溯。例如实现一个大的feature, 比较好的做法是多次分步实现提交，而不是多个feature在同一个commit里面提交。

33.

Git中的换行问题：

由于历史原因，不同的操作系统，在处理换行符时，使用了不同的方案。Windows 操作系统使用了 CRLF，而 Unix 阵营的操作系统则使用了 LF。Mac OS 最起初使用了 CR，后来到了 Mac OS X 后，改成了使用 LF，与 Unix 阵营保持了一致。虽然目前很多代码编辑器都支持自动识别和切换换行符风格。

Windows是CR（Carriage-Return，也就是\r）回车+LF(Line-Feed, 也就是\n) 换行

Mac和Linux是换行LF(Line-Feed)

git在维护版本库的时候统一使用的是LF

在Git中遇到的一个换行问题是， 如果不规定统一的换行符，那么文件在不同系统修改以后，在上传到repo中，会让我们diff时显示有变化，产生大量无必要的diff信息， 频繁的commit.

34.

git checkout -- <file> 用于还原工作区的文件， 指令先从缓存区中拉取版本还原替换工作区的文件，如果没有再到版本库中拉取还原代替工作区的文件。

35.

Git的基本两种开发模式

基于分支：

如果我们属于该项目，可以在该项目中创建其他分支，那么我们可以通过创建分支，然后Merge（Gitlab）或者Pull Request(Github)上把代码放入master中。

基于Fork模式：  
如果我们不属于将项目，可以通过Fork转到自己名下的库，然后进行开发，最后向项目进行Merge(Gitblab)或者Pull Request(Github)让项目管理者放入将项目的分支当中。

36.

Release分支



我们在项目开发的过程中，会通过不断的开发分支然后merge进入master.在某个时间或者某个feature完成以后，将会发布产品release.此时最好的做法在master分支上，创建一个Release分支并且以该分支上的节点创建release tag. 这个release分支，只会被让针对该release分支的fix-bug的分支merge进来，任何新的开发分支都不应该再merge进去此release分支。就如上图我们首先在master上创建release 1.x分支，然后经过开发，master不断融入新的feature分支代码，然后又创建release 2.x分支，然后继续迭代开发。当遇到release 1.x分支的bug出现时候，我们只需要在release 1.x分支上创建fix-bug分支，此次该fix-bug分支的代码环境时release 1.x的不包括以后迭代的其他开发代码，当我们修复完成后，merge进入release 1.x分支，tag下一个序号的该release 1.x代码完成修复工作。

37.

git cherry-pick可以理解为”挑拣”提交，它会获取某一个分支的单笔提交，并作为一个新的提交引入到你当前分支上。 当我们需要在本地合入其他分支的提交时，如果我们不想对整个分支进行合并，而是只想将某一次提交合入到本地当前分支上，那么就要使用git cherry-pick了

在所需要提交的分支上git cherry-pick [需要pick的提交号]

38.

任何将会改变git log的操作都不应该在master这些公共分支上进行。例如rebase, 或者git reset

39.

git push origin 不加分支名的默认行为

在git push origin在不加分知名的默认行为是由git配置中的push.default所规定的

如果push.default为matching, 则默认不加分支名进行push的时候，将会将本地**所有的**local branch push到远程仓库中对应匹配的分支

如果push.default为simple, 则默认不加分支名进行push的时候，将会将本地**所在的当前分支**local branch push到远程仓库中对应匹配的分支

Push\_default=matching 是Git1.x版本的默认行为， Push\_default=simple是Git2.x的默认行为。

如果我们想使用matching方式，可以在命令行输入：

git config --global push.default matching

1

如果我们想使用simple方式，可以在命令行输入：

git config --global push.default simple

40.

git remote show origin 可以展示本地库分支于远程分支的情况。可以看到有哪些远程分支本地不存在，或者有哪些本地分支在远端已经删除，以及哪些本地分支已经落后于远端库分支。

41.

git bash中如果中文无法显示：

git config --global core.quotepath false

42.

ssh -T 域名 用于测试是否是可以ssh域名成功。

若地址为：[git@orbw-git.ca.alcatel-lucent.com:xinde/regressionFlaskProjects.git](mailto:git@orbw-git.ca.alcatel-lucent.com:xinde/regressionFlaskProjects.git)

[则域名是orbw-git.ca.alcatel-lucent.com](mailto:则域名是git@orbw-git.ca.alcatel-lucent.com), 后面xinde/regressionFlaskProjects.git是用户以及所创建的项目名。

43.

通过在~/.ssh/下新建config文件, 为SSH连接进行配置，从而可以在同一台机器对**不同的域名的库**SSH连接时，自动使用不同的SSH密钥。

例如：#github配置

**Host github.com**

HostName github.com

IdentityFile ~/.ssh/github/id\_rsa --》此域名使用该密钥

User git

#gitoschina的配置

Host git.oschina.net

HostName git.oschina.net

IdentityFile ~/.ssh/id\_rsa --》此域名使用该密钥

如果之前有设置全局用户名和邮箱的话，需要unset一下

git config --global --unset user.name

git config --global --unset user.email

P. S SSH存放在~/.ssh下，所以如果我们以不同用户登陆那么~将会反映成不同用户，从而也会使用不同的SSH Key.

P. S 我们也可以在同一个用户下，为同一个用户指定不同域名的库使用不同的SSH Key.

44.

Git在本地，全局，系统级配置文件中，user.name, user.email是用于表明提交人的姓名，通过email关联到某个账户，计算活跃度的，显示在提交记录的头像名字而已。并不是验证是否可以access,是否可以push的功能，验证的功能在与SSH KEY以及https中输入的地址和对应的密码。

45.