|  |  |
| --- | --- |
| 文档编号：[pxxx - ]xxx - 1xx - ymd - xxxx | 说明（保持文档结构，内容多则加附件） |
| 规则：（[项目代码/编号-]文档类别/简介-组员编号-填写日期-流水号） [...]表示可选项 | |

**Git、GitHub、GitLab基本原理、安装与应用**

# Git基本原理

## Git和SVN对比

1、Git是分布式，SVN不支持分布式（集中式，分服务端和客户端）。

2、Git是元数据，SVN文件。

## Git概览

工作区：版本控制的目录

暂存区：.git/index

版本库：.git

## 远程仓库

GitHub、GitLab、码云

# 常用命令

|  |  |
| --- | --- |
| ***命令*** | ***简述*** |
| git init | 初始化，将当前目录作为工作目录 |
| git add | git add 文件|目录 ：提交指定的文件或目录到git暂存区  git add -A ：提交所有变化  git add -u ：提交被修改(modified)和被删除(deleted)文件，不包括新文件(new)  git add . ：提交新文件(new)和被修改(modified)文件，不包括被删除(deleted)文件 |
| git commit -m/-am | 将暂存区的文件添加到版本库 |
| git status | 查看暂存区的状态 |
| git diff | 查看对比：  （1）默认，不带参数，比较工作区与暂存区  （2）git diff --staged 或者git diff --cached [<path>...] ：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）  （3）git diff HEAD [<path>...] ：比较工作区与最新本地版本库，如果HEAD指向的是master分支，那么HEAD还可以换成master  （4）git diff commit-id [<path>...] ：比较工作区与指定commit-id的差异  （5）git diff --cached [<commit-id>] [<path>...] ：比较暂存区与指定commit-id的差异  （6）git diff [<commit-id>] [<commit-id>] ：比较两个commit-id之间的差异  （7）git diff > patchName ：将本仓库工作区的修改拷贝一份到其他机器上使用，但是修改的文件比较多，拷贝量比较大，此时我们可以将修改的代码做成补丁，之后在其他机器上对应目录下使用 git apply patch （git apply --check patch 检验补丁能否应用，如果没有任何输出表示可以顺利接受这个补丁，另外可以使用git apply --reject patch将能打的补丁先打上，有冲突的会生成.rej文件，此时可以找到这些文件进行手动打补丁）将补丁打上即可  git diff --cached > patchName //是将我们暂存区与版本库的差异做成补丁  git diff --HEAD > patchName //是将工作区与版本库的差异做成补丁  git diff fileName > patchName //将单个文件做成一个单独的补丁  （8）git diff branchName [<path>...] ：当前分支的文件与branchName 分支的文件进行比较  （9）git diff branchA branchB --full-index ：比较两个分支 |
| git reset HEAD | 撤销暂存区内容  git reset --hard HEAD ：回退到HEAD，本地所做的修改都无效，除了新增的文件和文件夹！！  git checkout文件|目录：恢复本地误删的文件或文件夹 |
| git rm | 删除 |
| git mv | 重命名 |
| git branch | git branch ：查看本地分支  git branch -a :查看本地及远程仓库的分支  git branch --all ：查看本地及远程仓库的分支  git branch [分支名称] ：创建指定分支，会将当前分支的文件复制到新创建的分支  git branch -d branchName ：删除分支(不能删除当前分支)  git branch -D branchName ：强制删除分支  git push origin --delete branchName ：删除远程分支 |
| git checkout | git checkout branchName ：切换到指定分支  git checkout --<file> ：先从缓存区中拉取版本还原，如果没有再到版本库中拉取还原 |
| git merge | git merge --no-ff -m "描述" branchName ：分支合并到主分支，若与主分支有冲突，分支可以通过git merge master来获取解决冲突后的内容 |
| git log | git log --graph --oneline ：显示简短的commit的SHA1与注释  git log -p ：显示补丁patchs |
| git tag | git tag -a v1.0.0 ：打版本信息，在弹出的会话框填写版本描述信息  git tag ：查看版本信息  git log --oneline --decorate --graph ：查看版本信息 |
| git remote | git remote -v ：查看远程链接 |

# GitHub

## GitHub注册

## GitHub仓库管理

### 新建仓库

.gitignore

### 删除仓库

首先登录github帐号，以远程仓库test为例，选择test，进入项目页面，右下方找到并点击settings，向下拉找到Delete this repositoty并点击，在弹出对话框中输入项目名称，最后点击下方按钮即可删除仓库。

# Git应用

## 关于文件忽略

|  |
| --- |
| ***# 删除.settings***  ***git>>*** git rm -r -n --cached .settings ***# 预览***  ***git>>*** git rm -r --cached .settings  ***git>>*** git commit -m "删除.settings"  ***git>>*** git push origin master  ***# 删除bin***  ***git>>*** git rm -r --cached bin  ***git>>*** git commit -m "删除bin"  ***git>>*** git push origin master  ***# 删除.classpath .project***  ***git>>*** git rm -r --cached .classpath  ***git>>*** git rm -r --cached .project  ***git>>*** git commit -m "删除.classpath和project"  ***git>>*** git push origin master  ***# 修改本地 .gitignore 文件 并提交***  ***git>>*** git add \*  ***git>>*** git commit -m "修改.gitignore"  ***git>>*** git push origin master |

## 多人协作开发

### 互联网环境

#### GitHub远程仓库

* **Git全局设置**

|  |
| --- |
| ***# 查看Git全局设置***  ***git>>*** git config --global --list  ***# egg 配置用户名和邮箱用于GitHub远程仓库的SSH免密***  ***git>>*** git config --global user.name "polarisgh"  ***git>>*** git config --global user.email "450733605@qq.com"  ***# egg SmartAI gitlab 账户密码***  ***git>>*** git config --global user.name "hewei"  ***git>>*** git config --global user.email "78645032@qq.com" |

* **配置GitHub SSH key**

生成SSH key脚本命令格式：ssh-keygen -t rsa -C "youremail@example.com"。

|  |
| --- |
| ***git>>*** *ssh-keygen -t rsa -C "450733605@qq.com"*  ***#用md5的方式查看指纹数据***  ***git>>*** *ssh-keygen -E md5 -lf /c/Users/Administrator/.ssh/id\_rsa* |

用户目录下（如：C:\Users\Administrator\.ssh）生成ssh文件夹，打开id\_rsa.pub，复制key，进入GitHub，Account => Settings（账户配置）=> SSH and GPG keys => New SSH key，填写标题后保存。

* **测试GitHub连通性**

|  |
| --- |
| ***# 验证是否可以连通github.com站点***  ***cmd>>*** *pint github.com*  ***# 如果不能连通需要加入目录（C:\Windows\System32\Drivers\etc\hosts）配置hosts文件***  ***# GitHub***  192.30.253.113 github.com  192.30.252.131 github.com  185.31.16.185 github.global.ssl.fastly.net  74.125.237.1 dl-ssl.google.com  173.194.127.200 groups.google.com  192.30.252.131 github.com  185.31.16.185 github.global.ssl.fastly.net  74.125.128.95 ajax.googleapis.com  ***# 配置完成后再次ping测试*** |

* **验证GitHub SSH**

|  |
| --- |
| ***git>>*** *eval $(ssh-agent -s)*  ***git>>*** *ssh-agent -s*  ***git>>*** *ssh-agent bash*  ***git>>*** *ssh-add ~/.ssh/id\_rsa*  ***# 验证，如果出现“Hi xxx! You've successfully authenticated, but GitHub does not provide shell access.”则没有问题***  ***git>>*** *ssh -T git@github.com* ***# ssh -vT git@github.com -i /c/Users/Administrator/.ssh/id\_rsa*** |

* **关联Git远程仓库**

（1）创建新的git本地仓库

|  |
| --- |
| ***git>>*** *git clone git@github.com:polarisgh/p0.git*  ***git>>*** *cd p0*  ***git>>*** *touch README1.md*  ***git>>*** *git add README1.md*  ***git>>*** *git commit -m "add README1"*  ***git>>*** *git push -u origin master* |

（2）上传本地已存在的工程目录

|  |
| --- |
| ***git>>*** *cd 已存在工程目录[p1]*  ***git>>*** *echo "# git-test p1" >> README.md*  ***git>>*** *git init*  ***# 本地仓库关联到GitHub仓库上***  ***git>>*** *git remote add origin git@github.com:polarisgh/p1.git*  ***git>>*** *git add .*  ***git>>*** *git commit -m "first commit"*  ***git>>*** *git push -u origin master* |

（3）上传本地已存在的git本地仓库

|  |
| --- |
| ***git>>*** *cd已存在git本地仓库[p2]*  ***# 本地仓库关联到GitHub仓库上***  ***git>>*** *git remote add origin git@github.com:polarisgh/p2.git*  ***git>>*** *git push -u origin --all*  ***git>>*** *git push -u origin --tags* |

参考：https://blog.csdn.net/u012467855/article/details/81284343

fork

#### GitLab远程仓库

#### 码云远程仓库

### 局域网环境

#### Git共享仓库

裸仓库

从裸仓库 clone 下来的本地仓库可以进行正常的 push 操作， 但是从一般仓库 clone 下来的本地仓库却不行。 这也正是裸仓库存在的意义。 裸仓库一般情况下是作为远端的中心仓库而存在的。

# 开始架设 Git 服务器前，需要先把现有仓库导出为裸仓库

$ git clone --bare my\_project my\_project.git

$ scp -r my\_project.git user@git.example.com:/opt/git

所有对该服务器有 SSH 访问权限，并可读取 /opt/git 目录的用户都可以用下面的命令克隆该项目：

$ git clone user@git.example.com:/opt/git/my\_project.git

【git将裸仓库移动到服务器】https://www.cnblogs.com/liyuanhong/articles/4928643.html

【git裸仓库简单自动化部署】https://www.jianshu.com/p/e6fe580a5e28

【git共享仓库多人协作开发】https://www.jianshu.com/p/54ad353b6cbb

#### 搭建GitLab服务器

## IDE集成

EGit

Git pull强制拉取并覆盖本地代码

git fetch --all

git reset --hard origin/master

git pull

# 问题

1.git branch查看不到任何分支。

***原因***：因为git的分支必须指向一个commit，没有任何commit就没有任何分支

提交第一个commit后git自动创建master分支

***解决方法***：使用git init命令创建仓库，执行git add . 和git commit（提交成功后）,再使用git branch命令，可显示出本地分支。（参考：https://blog.csdn.net/qq\_39671159/article/details/81261049）