Git_Foundation

> Based on 【GeekHour】一小时Git教程 [https://www.bilibili.com/video/BV1HM411377j/?share_source=copy_web&vd_source=8c8806bf56ee59fe86de67e00d08c354]

0_初始化设置

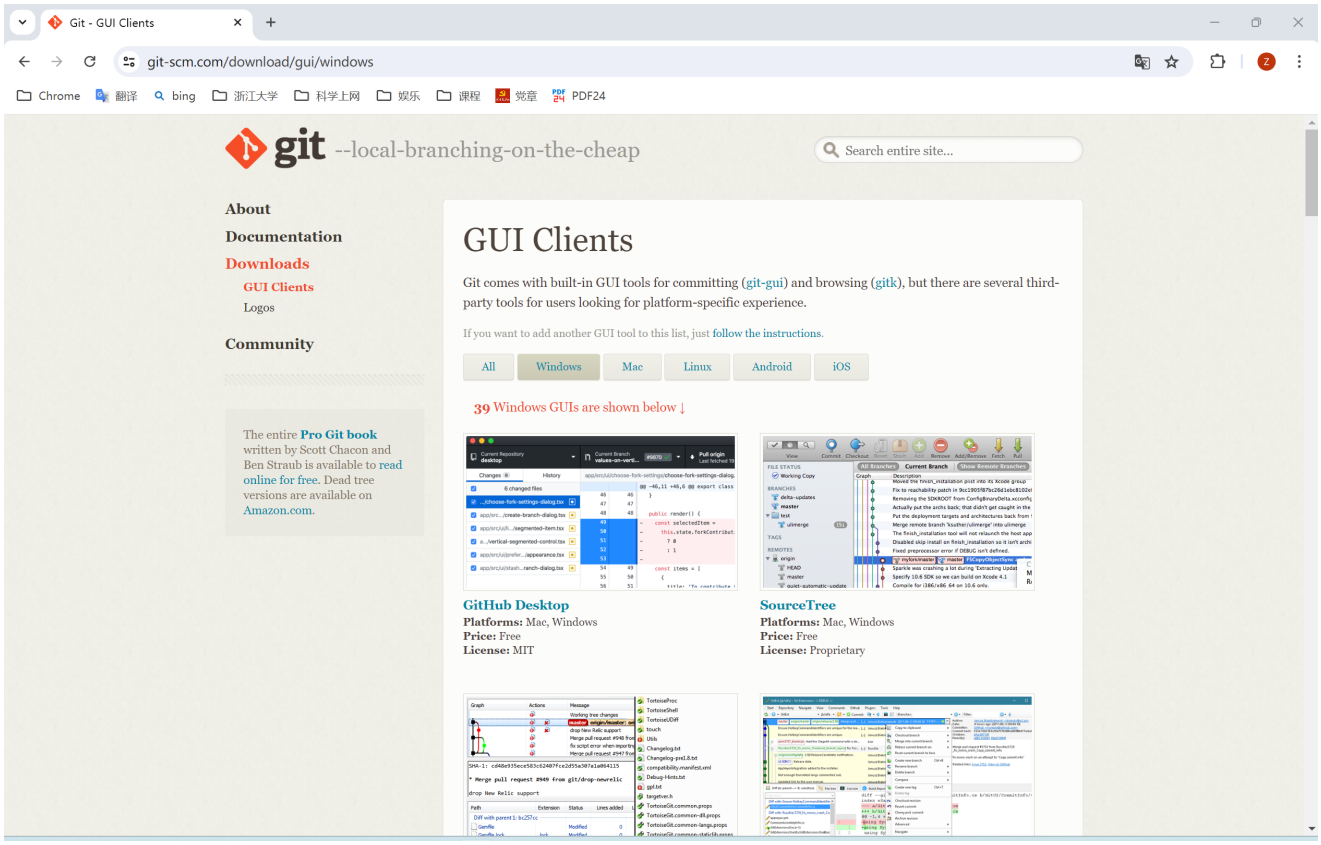
（1）Git_使用方法

---

1. 命令行

2. 图形化界面（GUI）

GUI Clients



> 推荐：SourceTree（免费），GitKraken（分免费和付费版本）

3. IDE 插件 / 扩展

> `git --version` or `git -v` 可以查看 Git 下载情况和版本信息

(2) 基础配置

Git 命令：

```
git config --global user.name "Right Or Not"
git config --global user.email jue1251665735@outlook.com
git config --global credential.helper store
git config --global init.defaultBranch main

# docs.github.com
git config --global core.editor "atom --wait"
git config --global core.editor "code --wait"
git config --global core.editor "subl -n -w"

git config --list
```

命令含义：

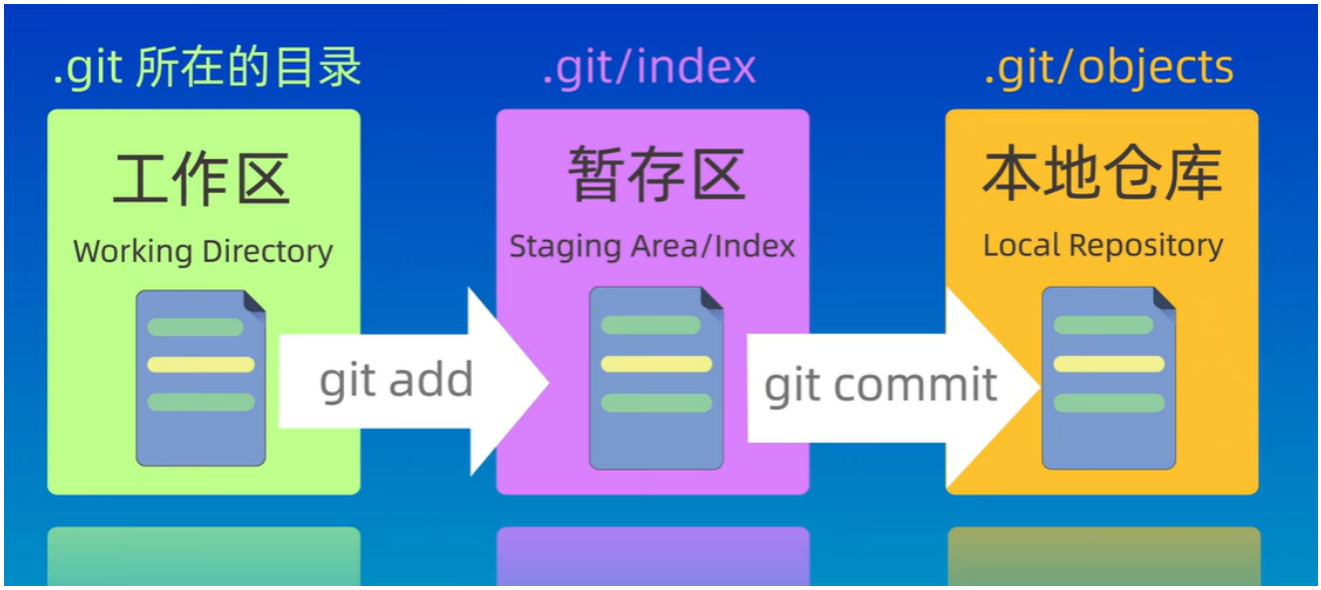
- 全局配置 Git 用户名为 Right Or Not
- 全局配置 Git 用户邮箱为 jue1251665735@outlook.com
- 全局配置 Git 记住用户名密码
- 全局修改主分支默认名称为 `main`
- 全局配置 Git 代码编辑器为 ATOM
- 全局配置 Git 代码编辑器为 VSCode
- 全局配置 Git 代码编辑器为 Sublime
- 查看 Git 配置

- `config` 参数：
  - `Local` ： 本地配置 只对本地仓库有效；
  - `--global` ： 全局配置 对所有仓库都有效；
  - `--system` ： 系统配置 对所有用户都有效。
- `Local` **一般可以省略** + `--system` **建议慎用** = **一般都使用** `--global`

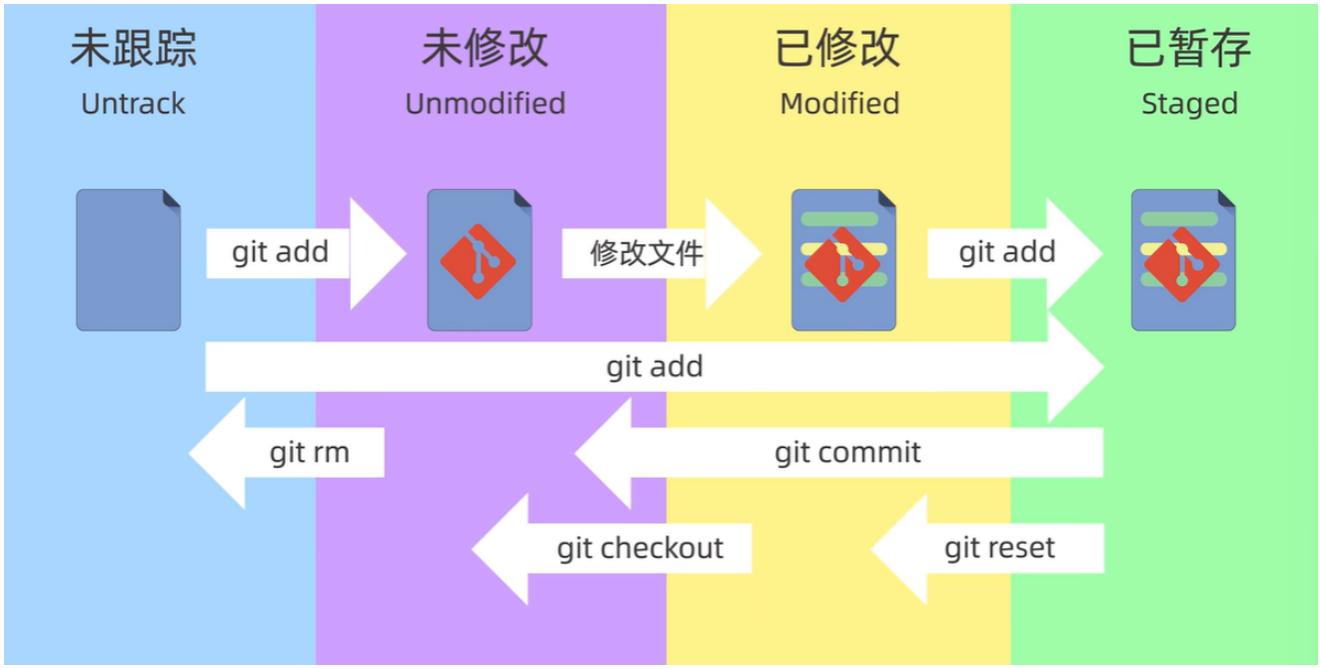
（3）工作区域 & 工作状态

---

1. 工作区域



2. 工作状态



1_本地仓库及基本操作 Repository

（1）新建本地仓库

Git 命令：

```
# git init
git init
git init my-repertory

# git clone
git clone <URL>

# git status
git status
git status -s
```

命令含义：

- 将当前文件夹初始化为一个空仓库
- 在当前文件夹中创建一个名为 `my-repertory` 的空仓库

- 从 `<URL>` 所对应的网页上克隆仓库（Github or Gitee）

  > 使用 `git clone <URL>` 的方式需要 `#配置SSH密钥`

- 查看仓库的状态
- 查看仓库的状态（简略模式）

（2）添加文件 git add

Git 命令：

```
# git add
git add <filename>
git add .
git add *.txt
git ls-files --stage
```

命令含义：

- 将文件 `filename` 添加到暂存区
- 将所有文件添加到暂存区
- 将所有 `.txt` 文件添加到暂存区
- 显示所有暂存区中的文件及其状态

（3）提交文件 git commit

Git 命令：

```
# git commit
git commit -m "message"
git commit -a -m "message"
git commit -am "message"
git status

# git log
git log
git log --oneline
git log --pretty=oneline
git log --oneline --graph --decorate  --all
```

命令含义：

- 提交暂存区的文件
- 将所有已修改的文件添加到暂存区并提交更改（被修改文件已经被追踪）
- 将所有已修改的文件添加到暂存区并提交更改（被修改文件已经被追踪）
- 查看仓库状态
- 查看提交记录
- 查看提交记录（简略模式）
- 查看提交记录（简略模式）
- 可视化地呈现所有分支及其关系

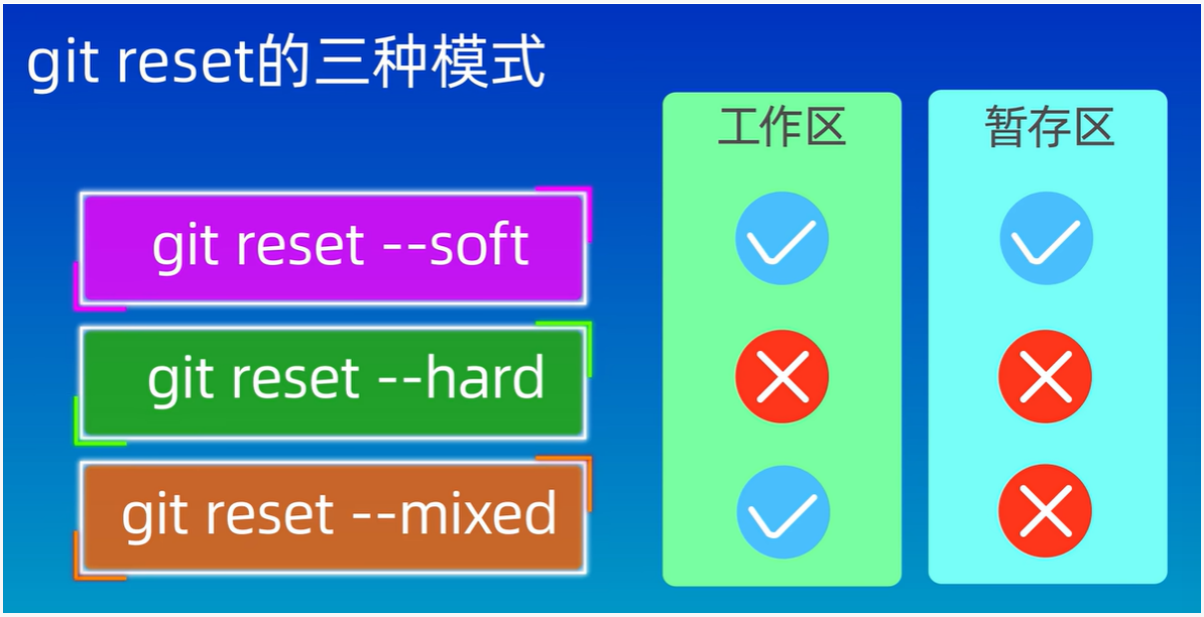
（4）版本回退 git reset

Git 命令：

```
# git reset
git reset <ID>
git reset HEAD^
git reset HEAD~
git reset --mixed <ID>
git reset --soft <ID>
git reset --hard <ID>
```

命令含义：

- 回退到 `ID` 对应的版本
- 回退到上一个版本
- 回退到上一个版本
- `mixed` 模式回退到 `ID` 对应的版本
- `soft` 模式回退到 `ID` 对应的版本
- `hard` 模式回退到 `ID` 对应的版本

> 三种不同模式的区别见下图：
>
> 
>

（5）文件删除 git rm

Git 命令：

```
# git rm
git rm <filename>
git rm --cached <filename>
git rm -r <foldername>
git commit -m "delete <filename>"
```

命令含义：

- 删除工作区和暂存区中的文件 `filename`
- 删除暂存区中的文件 `filename` （保留工作区中的文件）

- 递归删除 `foldername` 目录下的所有字母和文件
- 提交删除

> 删除后一定不要忘记提交删除！！！

2_分支及基本操作 Branch

Git 命令：

```
# Branch
git branch
git branch <branchname>
git switch <branchname>

# git merge
git switch main
git merge <branchname>
git branch -d <branchname>
git branch -D <branchname>

git merge --quit
git merge --abort

# git rebase
git switch dev
git rebase <branchname>
```

命令含义：

 # Branch

- 查看分支
- 新建名为 `branchname` 的分支
- 切换到名为 `branchname` 的分支上

 # git merge

- 切换到 `main` 分支之上
- 将名为 `branchname` 的分支合并到 `main` 分支（必须在 `main` 分支上操作）
- 删除已经合并的名为 `branchname` 的分支
- 强制删除名为 `branchname` 的分支
- 合并冲突时中断合并（保留已经进行的合并步骤）
- 合并冲突时终止合并（强制撤销所有已经进行的合并步骤）

 # git rebase

- 切换到 `dev` 分支之上
- 将 `dev` 分支嫁接到名为 `branchname` 的分支之上

> `git merge` 和 `git rebase` 优缺点比较：
>
> - `git merge`
> - 优点：不会破坏原有分支的提交记录
> - 缺点：会额外产生提交节点 分支图比较复杂
> - `git rebase`
> - 优点：提交历史呈线性 直观、简洁
> - 缺点：改变了原有分支的提交历史

> 习惯性采用的分支名称：
>
> - `main` ： 主分支（稳定的发布版本）
> - `dev` ： 副分支（长期存在的分支 集成所有新功能）
> - `feat` ： 特殊功能开发分支（feature）
> - `bugfix` ： bug 修复分支
> - `release` ： 新版本准备分支（一般在 `dev` 分支之下 用于新版本发布前最后的测试和调试）
> - `docs` ： 文档更新分支

3_查看差异 git diff

Git 命令：

```
# git diff
git diff
git diff <ID>
git diff --cached

git diff <ID_1> <ID_2>
git diff HEAD~ HEAD
git diff HEAD~n HEAD
git diff HEAD^ HEAD
git diff HEAD^n HEAD

git diff <ID_1> <ID_2> <filename>

git diff <branchname_1> <branchname_2>
```

命令含义：

- 查看工作区和暂存区之间的差异
- 查看工作区和 `ID` 对应的版本之间的差异
- 查看暂存区和版本库之间的差异
- 查看 `ID_1` 对应的版本和 `ID_2` 对应的版本之间的差异
- 查看当前版本和上一次提交的版本之间的差异
- 查看当前版本和上 `n` 次提交的版本之间的差异
- 查看当前版本和上一次提交的版本之间的差异
- 查看当前版本和上 `n` 次提交的版本之间的差异
- 查看 `ID_1` 对应的版本和 `ID_2` 对应的版本之间 `filename` 文件的差异
- 查看 `branchname_1` 对应分支和 `branchname_2` 对应分支之间的差异

4_配置SSH密钥

#配置SSH密钥

```
# 打开终端（Terminal）
Win + R: cmd Enter
Win + S: cmd Enter

# 切换到主目录
cd C:\Users\username
// 一般情况下打开终端（Terminal）默认在主目录下

# 创建 .ssh 文件夹
mkdir .ssh

# 设置 .ssh 文件夹权限（可选）
chmod 700 .ssh

# 创建 SSH 密钥
cd .ssh
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
// -C 后的内容表示添加一个注释（可选）

# 查看公钥内容
cat <id_rsa>.pub
// 将 .pub 文件中的公钥内容复制粘贴到 GitHub 的 Setting 之中完成配置
```