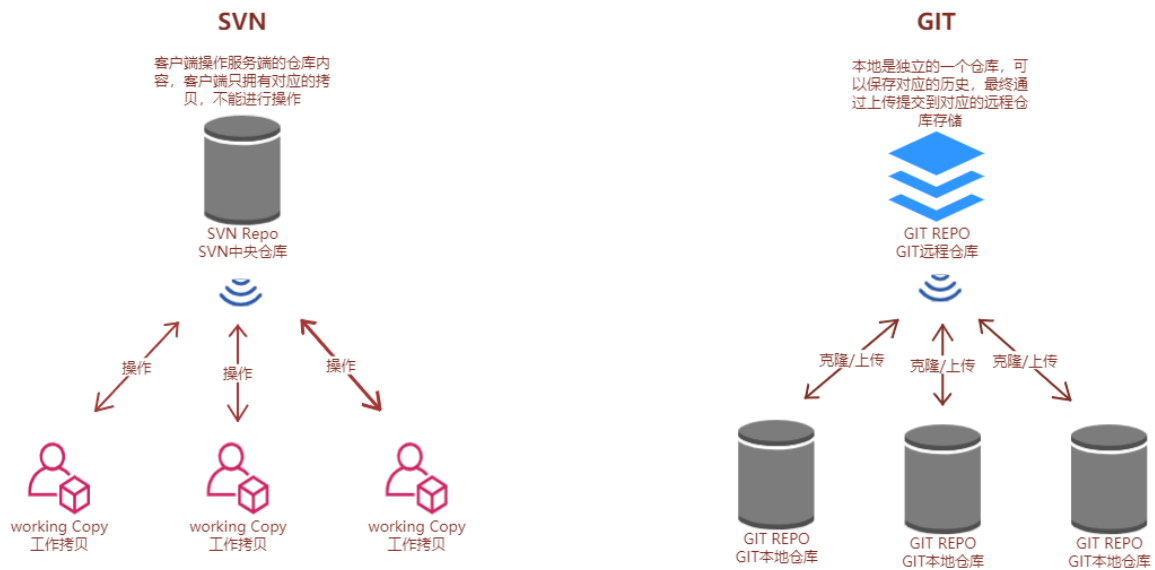


GIT

概述

git是一个版本管理工具，它是用于管理对应的代码的版本的。它是一个集中式的代码管理工具（支持分布式）。相同的软件还有svn（集中式版本管理工具，它不具备分布式的功能）。

svn及git的区别




了解git及svn的区别

GIT环境搭建

[git官网地址](#)

- 下载git

The screenshot shows the Git website homepage. At the top, the Git logo is followed by the tagline '--distributed-is-the-new-centralized'. A search bar is located on the right. The main content area features two paragraphs of text: 'Git is a **free and open source** distributed version control system designed to handle everything from small to very large projects with speed and efficiency.' and 'Git is **easy to learn** and has a **tiny footprint** with **lightning fast performance**. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like **cheap local branching**, convenient staging areas, and multiple workflows.' To the right of the text is a 3D illustration of a branching structure. Below the main content, there are four sections: 'About' (The advantages of Git compared to other source control systems.), 'Documentation' (Command reference pages, Pro Git book content, videos and other material.), 'Downloads' (GUI clients and binary releases for all major platforms.), and 'Community' (Get involved! Bug reporting, mailing list, chat, development and more.). On the right side, there is a monitor displaying the 'Latest source Release 2.40.0' and a 'Download for Windows' button, with a link '点击进去下载' (Click to download) next to it.

 **git** --distributed-even-if-your-workflow-isnt

Search entire site...

About
Documentation
Downloads
GUI Clients
Logos
Community

The entire **Pro Git book** written by Scott Chacon and Ben Straub is available to [read online for free](#). Dead tree versions are available on [Amazon.com](#).

Download for Windows

[Click here to download](#) the latest **(2.40.0)** **64-bit** version of **Git for Windows**. This is the most recent **maintained build**. It was released **2 days ago**, on 2023-03-14.

Other Git for Windows downloads

Standalone Installer → window版本下载

[32-bit Git for Windows Setup.](#)

[64-bit Git for Windows Setup.](#)

Portable ("thumbdrive edition")

[32-bit Git for Windows Portable.](#)

[64-bit Git for Windows Portable.](#)

Using winget tool

Install **winget** tool if you don't already have it, then type this command in command prompt or Powershell.

```
winget install --id Git.Git -e --source winget
```

The current source code release is version **2.40.0**. If you want the newer version, you can build it from [the source code](#).

Now What?

Now that you have downloaded Git, it's time to start using it.

- 安装git

无脑安装

- 测试（出现了对应 git gui（ui视图） 以及 git bash（指令））

wind+r 进入 cmd

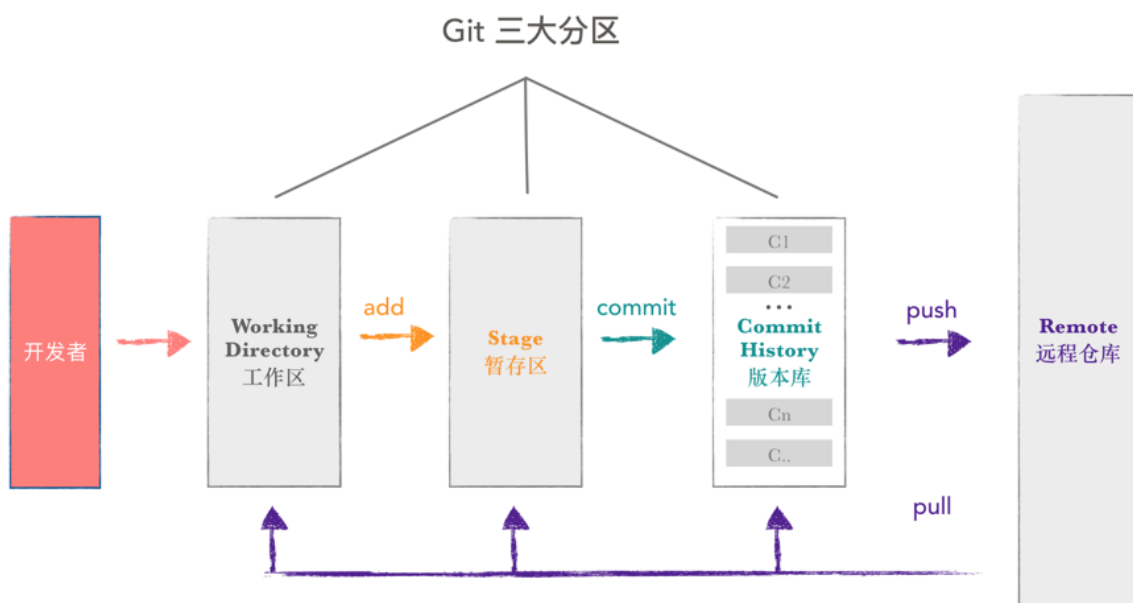
```
git --version #查看版本
```

```
C:\Users\29769>git --version
git version 2.35.1.windows.2
```



git的分区

- 工作区（代码书写）
- 暂存区（暂存对应的文件）
- 版本库（历史区 记录所有的提交历史）



git的相关操作

指令化操作

git bash 里面是对应的linux指令区，支持linux指令

常见的linux指令

- cd 进入某个文件夹
- ls 查看当前的文件
- shutdown 关机
- reboot 重启
- tar 解压
- vi 编辑器（编辑文件）
- rm 删除
- mv 移动
- clear 清空控制台
- ...

```
ls
```

创建本地仓库

```
git init
```

```
29769@LAPTOP-8D5355PJ MINGW64 /d/CSH5/CSH5-2210/day29/code
$ git init
Initialized empty Git repository in D:/CSH5/CSH5-2210/day29/code/.git/
```

创建一个隐藏的文件夹 .git文件夹（本地仓库）

 .git	2023/3/16 12:00	文件夹
--	-----------------	-----

暂存区操作

加入到暂存区

存入文件到暂存区

```
git add 文件
```

```
29769@LAPTOP-8D5355PJ MINGW64 /d/CSH5/CSH5-2210/day29/code (master)
$ git add ./b.txt
```

查看状态

```
git status
```

```
$ git status
On branch master
No commits yet
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   b.txt
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    a/
```



添加文件夹（添加文件夹的下的所有的文件）

```
git add 文件夹路径
```

```
29769@LAPTOP-8D5355PJ MINGW64 /d/CSH5/CSH5-2210/day29/code (master)
$ git add a

29769@LAPTOP-8D5355PJ MINGW64 /d/CSH5/CSH5-2210/day29/code (master)
$ git status
On branch master
No commits yet
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   a/a.txt
    new file:   b.txt
```

添加所有的文件

```
git add *
git add .
git add --all
```

```
29769@LAPTOP-8D5355PJ MINGW64 /d/CSH5/CSH5-2210/day29/code (master)
$ git add *
```

```
29769@LAPTOP-8D5355PJ MINGW64 /d/CSH5/CSH5-2210/day29/code (master)
$ git add --all
```

从暂存区撤回

- 不会影响工作区

```
git reset HEAD -- 文件名
git reset HEAD -- 文件夹名
git reset HEAD -- * #撤回所有
git reset HEAD -- . #撤回所有
```

```

29769@LAPTOP-8D5355PJ MINGW64 /d/CSH5/CSH5-2210/day29/code (master)
$ git reset HEAD -- *

29769@LAPTOP-8D5355PJ MINGW64 /d/CSH5/CSH5-2210/day29/code (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        a.bmp
        a.docx
        a/
        b.rtf
        b.txt
        c.rtf

nothing added to commit but untracked files present (use "git add" to track)

```

版本库操作

从暂存区提交到版本库 (暂存区就没有了)

如果是第一次进行提交操作需要设置对应的用户名及邮箱号

```

git config user.username 名字 --global
git config user.email 邮箱 --global

```

提交指令进入vi编辑器

```

git commit 文件
git commit 文件夹
git commit *
git commit .

```

vim 编辑器模式

- 阅读模式 (进入就是阅读模式)
- 插入模式 (在阅读模式中 按下 a 或 i 或 o 进入对应的插入模式 返回阅读模式按下esc键)
- 命令行模式 (通过阅读模式进入命令行模式 :wq (保存后退出))

```

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   new file:   a.bmp
#
# Untracked files:
#   a.docx
#   a/
#   b.rtf
#   b.txt
#   c.rtf
#
~
~
~

```

```
29769@LAPTOP-8D5355PJ MINGW64 /d/CSH5/CSH5-2210/day29/code (master)
$ git commit a.bmp
[master (root-commit) e7360cc] 这个是一个新的提交记录，里面添加了一个新的文件a.bmp
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 a.bmp
```

提交指定对应的信息

```
git commit 文件 -m 信息
```

```
29769@LAPTOP-8D5355PJ MINGW64 /d/CSH5/CSH5-2210/day29/code (master)
$ git commit a -m '提交了a文件夹'
[master 41de9bd] 提交了a文件夹
2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 a/a.ppt
create mode 100644 a/a.txt

29769@LAPTOP-8D5355PJ MINGW64 /d/CSH5/CSH5-2210/day29/code (master)
$ git log
commit 41de9bd79b49fe9d646c8cb9bece0dac4ec7a0c5 (HEAD -> master)
Author: 王某某 <7782740+mr_wang_oo@user.noreply.gitee.com>
Date: Thu Mar 16 15:04:03 2023 +0800

    提交了a文件夹

commit e7360cc9368fccbd39d72a8aa648428449d8582e
Author: 王某某 <7782740+mr_wang_oo@user.noreply.gitee.com>
Date: Thu Mar 16 14:48:24 2023 +0800

    这个是一个新的提交记录，里面添加了一个新的文件a.bmp
```

查看提交记录

```
git log
```

```
$ git log
commit e7360cc9368fccbd39d72a8aa648428449d8582e (HEAD -> master)
Author: 王某某 <7782740+mr_wang_oo@user.noreply.gitee.com>
Date: Thu Mar 16 14:48:24 2023 +0800

    这个是一个新的提交记录，里面添加了一个新的文件a.bmp
```

当前最新提交记录的分支 提交信息

从版本库撤回

```
git reset --hard 版本号
git reset --hard HEAD #撤回最新一次的提交
```

	是否影响工作区	是否影响暂存区
--soft	√	×
--mixed (默认的)	×	√
--hard	√	√

```
$ git reset --hard HEAD
HEAD is now at cc68cb9 第三次提交

29769@LAPTOP-8D5355PJ MINGW64 /d/CSH5/CSH5-2210/day29/code (master)
$ git reset --hard e7360cc9368fccbd39d72a8aa648428449d8582e
HEAD is now at e7360cc 这个是一个新的提交记录，里面添加了一个新的文件a.bmp
```

分支

分支是对应的git的主要组成，git通过对应的分支操作来进行功能的合并和对应的冲突的解决。分支相当于将对应的功能进行拆分每个子功能单独开辟一个分支，这样就可以让两个分支之间的耦合和冲突就减少了。

分支的命名规范

git 分支分为集成分支、功能分支和修复分支，分别命名为 master、feature 和 fix，均为单数。不可使用 features、future、hotfixes、hotfixs 等错误名称。

- master（主分支，永远是可用的稳定版本，**不能直接在该分支上开发**）
- master_check（未上线前的开发分支，该分支只做只合并操作，不能直接在该分支上开发，前期开发完成后将feature分支合并到此分支）
- online（线上分支，由发版人员确认测试没问题后，将online_check分支合并到此分支）
- develop 开发主分支
- online_check（开发主分支，所有新功能以这个分支来创建自己的开发分支，该分支只做只合并操作，不能直接在该分支上开发）
- feature-xxx（功能开发分支，在develop上创建分支，以自己开发功能模块命名，功能测试正常后合并到develop分支，开发完成后合并到online_check分支上）

```
feature/siliwu_querySchdule
```

- fix-xxx（修改bug分支，在master分支上创建，修复完成后合并到 online_check）

注意事项：

- 一个分支尽量开发一个功能模块，不要多个功能模块在一个分支上开发。
- feature 分支在申请合并之前，最好是先 pull 一下master_check分支下来，看一下有没有冲突，如果有就先解决冲突后再申请合并。

分支的创建

```
git branch 分支名
```

分支的查询（*表示当前分支）

```
git branch
```

```
$ git branch
devloop
* master
```

分支的切换

```
git checkout 分支名
```



```
$ git checkout devloop
Switched to branch 'devloop'
```

分支的删除（不能在当前分支内删除当前分支）

```
git branch -d 分支名
git branch -D 分支名
```

```
$ git branch -d a
Deleted branch a (was e7360cc).
```

分支合并

会产生一个新的版本记录

将对应的分支合并到当前分支

```
git merge 分支
```

```
$ git merge a
Updating e7360cc..de5c559
Fast-forward
 b.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 b.txt
```

衍合分支

```
git rebase 分支
```

```
$ git rebase b
Successfully rebased and updated refs/heads/devloop.
```

Git EE

概述：

git ee（码云 国内）是一个代码托管平台（远程仓库），它主要是用于托管对应的代码。与之相关的平台还有Git hub（国际）。

主要代码托管

- gitEE 国内的代码托管
- gitHub 国际的代码托管（很多的框架的代码）
- 公司内部如果使用的是git进行对应的版本管理那么会使用**git lab**作为远程仓库（它是自己内部搭建的（私服））

gitEE仓库创建

- 注册、登录



- 新建仓库



新建仓库

在其他网站已经有仓库了吗? [点击导入](#)

仓库名称 *

归属

王某某

路径 *

自动生成路径

仓库介绍

0/100

用简短的语言来描述一下吧

描述

- ☐ 开源 (所有人可见) ?
- ☒ 私有 (仅仓库成员可见)
- ☐ 企业内部开源 (仅企业成员可见) ?

- ☐ 初始化仓库 (设置语言、.gitignore、开源许可证)
- ☐ 设置模板 (添加 README、Issue、Pull Request 模板文件)
- ☐ 选择分支模型 (仓库创建后将根据所选模型创建分支)

自动创建.git文件夹

创建

第一种情况不设置对应的内容

拷贝对应的路径

在本地创建对应的git仓库 连接对应的远程仓库

```
git init
git remote add origin url #远程连接建立 第一次需要输入密码 (一次)
#数据准备 本地仓库操作
git add .
git commit .
#推送到对应远程仓库
git push origin 分支名
```

```
29769@LAPTOP-8D5355PJ MINGW64 /d/CSH5/CSH5-2210/day29/code/repo (master)
$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 282 bytes | 282.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote: Powered by GITEE.COM [GNK-6.4]
To https://gitee.com/mr_wang_oo/cs-2210c.git
912b900..4aef752 master -> master
```

第二种方法内容已经存在

拷贝路径

```
git clone url #克隆对应的内容
#本地仓库操作
git add .
git commit .
#推送到远程仓库
git push origin 分支名
```

相关指令

- git pull 拉取最新的数据（默认会进行合并的操作）
- git fetch 拉取数据（不会进行合并操作）
- git remote add origin url 建立连接
- git push origin 分支名 推送到远程仓库
- git diff 比对对应的分支之间的差异
- git clone 克隆对应的远程仓库内容

对应的冲突解决

对应的两个人在完成一个功能的时候 操作同一个分支中的内容，第一个版本号为0，a先提交 那么对应的远程仓库的版本就是1 如果b再进行提交它的版本是0 是小于远程仓库的版本的，这个时候就会产生冲突。需要人工介入。

解决冲突的方式

先拉取最新的再进行对应的操作（是保留谁的内容）

git pull 会合并对应的内容

使用命令行

```
git fetch origin master:hello
#再进行比对 出现差异
git diff hello
#人工干预
#再合并
git merge hello
```

使用vscode来解决



