



湖南工商大学
HUNAN UNIVERSITY OF TECHNOLOGY AND BUSINESS

《通用工具操作实训》

实训报告

姓 名:	张舒婷
学 号:	2409050079
专 业:	网络空间安全
班 级:	网安 2402
指导教师:	王海东
职 称:	讲师

前沿交叉学院

2025 年 5 月

《通用工具操作实训》评审表

姓名				学 号					
				专业班级					
实训内容									
评审意见	项目	具体要求	单项分值	分项评分分数参考区间					得分
				A	B	C	D	E	
	平时表现	学习态度认真，自主学习相关方法、总结归纳相关技术的能力优秀。积极与团队成员合作，积极参与讨论分析，团队关系融洽，能按时完成团队分配的工作。明确团队角色和分工，积极与团队成员合作，团队工作进展顺利。	20	18	16	14	12	<12	
	实操成绩	Git 版本控制(15 分):基础操作(5 分) 分支管理(5 分) 远程协作(5 分)； Docker (15 分)：镜像操作(5 分)，容器管理(5 分)，数据与编排(5 分)；攻防工具选修(20 分, 任选 1 项完成)。	50	45	40	35	30	<30	
	文档质量	文档规范，结构合理，文献工作量饱满，格式正确。	10	9	8	7	6	<6	
		实验原理正确，步骤完整，实验结果充分，结论有效。方案合理可行，效果好，有创新意识。工作量饱满，完成质量高。	20	18	16	14	12	<12	
	评审成绩： <input type="checkbox"/> 优秀（100-90） <input type="checkbox"/> 良好（89-80） <input type="checkbox"/> 中等（79-70） <input type="checkbox"/> 及格（69-60） <input type="checkbox"/> 不及格（<60）								
指导教师 签名		职称		时间	年 月 日				

目录

- 摘要..... 4
- 第 1 章 绪论..... 5
 - 1.1 实训背景与意义..... 5
 - 1.2 实训内容..... 5
- 第 2 章 git..... 5
 - 2.1 基本原理..... 5
 - 2.2 实验步骤..... 6
 - 2.3 实验结果与分析..... 9
- 第 3 章 Docker..... 9
 - 3.1 基本原理..... 9
 - 3.2 实验步骤..... 9
 - 3.3 实验结果与分析..... 11
- 第 4 章 Wireshark..... 12
 - 4.1 基本原理..... 12
 - 4.2 实验步骤..... 12
 - 4.3 实验结果与分析..... 15
- 第 5 章 总结..... 16

摘要

本实验报告旨在总结和记录在通用工具操作实训中的学习内容和实践经验。实训内容主要覆盖开发和网络攻防通用工具两个方向，包括 Git 的掌握、Docker 的掌握、网络协议分析工具（如 Wireshark）的使用。通过本次实训掌握了这些工具的基本操作和高级功能，提升了在实际项目中的应用能力，为后续的软件开发和网络安全工作奠定了坚实基础。

关键词：Git, Docker, Wireshark, 实训

第 1 章 绪论

1.1 实训背景与意义

随着信息技术的快速发展，软件开发、网络通信和安全防护等领域对高效工具的需求日益增长。掌握通用工具的操作技能不仅能提高工作效率，还能为复杂问题的解决提供技术支持。本次实训旨在通过实际操作，帮助学生熟练掌握 Git、Docker 和 Wireshark 等工具的使用方法，培养其在实际项目中的工具应用能力和问题解决能力。这些技能对于从事软件开发、网络安全和系统维护等职业的学生具有重要的实践意义。

1.2 实训内容

本次实训主要围绕现代软件开发中的关键技术展开，重点包括 Git 版本控制、Docker 容器化技术的学习与实践和 wireshark 的抓包操作的掌握。在 Git 部分，实训内容涵盖版本管理的基本原理、常用命令操作、分支管理策略以及团队协作开发流程，旨在帮助掌握高效的代码版本控制方法。Docker 部分则涉及容器化技术的核心概念、镜像构建与管理、容器部署及网络配置，通过实际案例练习，培养应用容器化的部署能力。此外，实训还深入网络分析领域，借助 Wireshark 工具开展网络抓包实验，包括协议解析、流量监控及故障诊断等实践环节，使学员能够全面理解网络通信机制。实训采用理论讲解与动手实操相结合的方式，通过阶段性任务和团队协作项目，逐步提升学员的技术实践能力和问题解决能力，最终完成可交付的代码仓库、容器化应用及完整的实训报告。

第 2 章 git

2.1 基本原理

在现代软件开发的复杂协作场景中，Git 作为一款卓越的分布式版本控制系统，其核心竞争力源于对工作目录、暂存区、本地仓库及远程仓库这四个关键区域的精妙设计与协同运作。这四个区域相互衔接又各司其职，共同构建起一套高效、灵活且可靠的代码管理体系，为开发者提供了从个体编辑到团队协作的全流程支持。其中，工作目录作为开发者直接接触的“前线阵地”，是代码文件实际存在并进行编辑的物理空间。在这里，每一个文件的创建、修改或删除都会被 Git 实时感知而已被管理的文件若发生变动，则会标记为“已修改”，清晰呈现与最近一次提交版本的差异；暂存区扮演着“筛选与准备”的过渡角色，它像一个精心设计的缓冲区，承接工作目录中的变更并为提交操作做好铺垫。作为过渡区域，它通过 `git add` 命令收纳工作目录的变更，为提交做准备；本地仓库存储完整版本历史，`git commit` 命令将暂存区内容固化为版本记录；而远程仓库作为团队协作的“中央枢纽”，通常部署在服务器上（如 GitHub、GitLab 等平台），是多人共享代码、同步进度的核心载体。它用于多人协作时的代码同步，借助

git push、git pull 等命令，实现本地与远程仓库的交互，支持并行开发与版本回溯。

2.2 实验步骤

创建一个新的文件夹并进入：

```
wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~ (master)
$ mkdir myproject
mkdir: cannot create directory 'myproject': File exists

wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~ (master)
$ cd myproject
```

初始化 git 仓库，使用 git add 将其加入暂存区，再通过 git commit 提交到本地仓库：

```
wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (master)
$ git init
Reinitialized existing Git repository in C:/Users/wandering Cloud/myproject/.git/

wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (master)
$ echo "#My Project" > README.md

wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (master)
$ git add README.md
warning: in the working copy of 'README.md', LF will be replaced by CRLF the next time Git touches it

wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (master)
$ git commit -m "Initial commit"
[master (root-commit) c3773e7] Initial commit
1 file changed, 1 insertion(+)
create mode 100644 README.md
```

用 git log 查看提交历史

```
wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (master)
$ git log --oneline
c3773e7 (HEAD -> master) Initial commit
```

```
wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (master)
$ echo -e

wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (master)
$ find .git/objects -type f |while read obj; do
> echo "对象:${obj}"
> git cat-file -p $(basename $(dirname $obj))$(basename $obj)
> done
对象: .git/objects/03/8572e7ca39e6ea9bf1067b261d41fc264666a7
#My Project
对象: .git/objects/27/42d5e490d58603b1a21c68423ff6bfcd83402d
100644 blob 038572e7ca39e6ea9bf1067b261d41fc264666a7 README.md
对象: .git/objects/c3/773e758b9919c32f97304077a1bf67d9da2e8a
tree 2742d5e490d58603b1a21c68423ff6bfcd83402d
author 001mby <233246976@qq.com> 1751371594 +0800
committer 001mby <233246976@qq.com> 1751371594 +0800
```

执行 git commit 时，master 分支如何更新

```
wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (master)
$ cat .git/refs/heads/master
c3773e758b9919c32f97304077a1bf67d9da2e8a
```

通过 git remote 命令添加远程仓库，git remote -v 查看参考信息

```
Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (master)
$ git remote add oringin https://github.com.001mby/myproject.git

Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (master)
$ git remote -v
oringin https://github.com.001mby/myproject.git (fetch)
oringin https://github.com.001mby/myproject.git (push)
```

首先查看当前分支（确认是否是 master）将本地 master 分支推送到远程 origin 仓库 再用 git push 命令将本地版本库的提交推

```
Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (master)
$ git branch
* master

Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (master)
$ git push -u origin master
fatal: 'origin' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.

Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (master)
$ git push
fatal: The current branch master has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream oringin master

To have this happen automatically for branches without a tracking
upstream, see 'push.autoSetupRemote' in 'git help config'.
```

完成从本地推送到远程，从远程拉取更新的完整流程

```
$ git fetch origin
fatal: 'origin' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.

Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (master)
$ git pull origin master
fatal: 'origin' does not appear to be a git repository
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.
```

URL 分别为 HTTPS 和 SSH 两种格式的远程仓库

```
Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (master)
$ git remote add origin https://github.com/001mby/myproject.git

Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (master)
$ git remote add origin git@github.com:001mby/myproject.git
error: remote origin already exists.
```

(3)

使用 git checkout -b dev 创建并切换到新分支 dev, 用 git branch 查看当前分支（带*的为当前分支）

```
Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (dev)
$ git branch -r

Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (dev)
$ git branch -a
* dev
  master
```



```
Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (master)
$ git checkout -b dev
Switched to a new branch 'dev'

Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (dev)
$ git branch
* dev
  master
```

在 dev 分支添加文件

```
Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (dev)
$ echo "这里是 dev 分支的内容" > dev_README.md
$ git add dev_README.md
warning: in the working copy of 'dev_README.md', LF will be replaced by CRLF the next time Git touches it
Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (dev)
$ git commit -m "在 dev 分支添加 README 文件"
$ git checkout master
```

用 git merge 命令将 dev 分支合并到 main 分支，并查看文件

```
Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (dev)
$ git merge dev
Already up to date.

Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (dev)
$ ls
README.md  dev_README.md

Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (dev)
$ cat dev_README.md
这里是 dev 分支的内容
```

模拟并解决合并冲突。在 main 中修改文件，切换到 dev 分支并修改同一文件，切换回 main 分支尝试合并，出现冲突，手动编辑 conflict_README.md 解决冲突

```
Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (dev)
$ echo "main 分支的修改" >> conflict_README.md
Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (dev)
$ git add conflict_README.md
warning: in the working copy of 'conflict_README.md', LF will be replaced by CRLF the next time Git touches it
Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (dev)
$ git commit -m
error: switch '-m' requires a value
Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (dev)
$ git commit -m "main 分支的修改"
(dev 239a41b) main 分支的修改
1 file changed, 1 insertion(+)
create mode 100644 conflict_README.md
```

```
Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (dev)
$ git checkout master
Switched to branch 'master'

Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (master)
$ git merge dev
Updating c3773e7..6071764
Fast-forward
 conflict_README.md | 2 ++
 dev_README.md      | 1 +
 2 files changed, 3 insertions(+)
 create mode 100644 conflict_README.md
 create mode 100644 dev_README.md

Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (master)
$ git add conflict_README.md

Wandering Cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (master)
$ git commit -m "解决合并冲突"
On branch master
Untracked files:
```



```
wandering cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (dev)
$ git checkout dev
Already on 'dev'

wandering cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (dev)
$ echo "dev分支的修改" >>conflict_README.md

wandering cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (dev)
$ git add conflict_README.md
warning: in the working copy of 'conflict_README.md', LF will be replaced by CRLF the next time Git touches it

wandering cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (dev)
$ git commit -m "dev分支的修改"
[dev 6071764] dev分支的修改
1 file changed, 1 insertion(+)

wandering cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (master)
$ git branch -d dev
Deleted branch dev (was 6071764).

wandering cloud@DESKTOP-IMH8HM6 MINGW64 ~/myproject (master)
$ git branch -D master
```

2.3 实验结果与分析

通过对 Git 实操，我清晰看到工作目录、暂存区、本地仓库的交互。新建文件是“未跟踪”状态，采用 `git add` 能把修改放到暂存区，再执行 `git commit -m` 提交到本地仓库，借助 `git log` 能查看提交历史。清晰的展现了文件在三个区域的状态变化。在远程仓库交互时，分别采用 `git push` 把本地代码推上去，`git pull` 拉取更新和 `git remote` 配置、查看远程仓库。在分支管理里，使用 `git checkout -b` 新建并切换分支，不同分支开发互不影响。通过 `git branch` 系列命令查看分支，在多分支并行开发，体验切换时工作目录变化。合并分支用 `git merge`，遇到冲突就手动改文件，再 `git add`，`git commit` 解决。Git 让我学会版本控制，团队开发时能管好代码，还能回退历史版本。

第3章 Docker

3.1 基本原理

Docker 是基于操作系统级虚拟化的容器化技术，核心靠镜像、容器、仓库三大组件协同工作。其中，镜像为只读模板，含应用及所有依赖，采用分层存储，可增量构建和共享复用，每层对应 Dockerfile 一条指令。容器是镜像的运行实例，通过添加可写层实现修改隔离，用写时复制机制保证轻量，秒级启动。借助 namespace 隔离资源，cgroups 限制 CPU、内存等，实现多容器并行不干扰。仓库用于集中存储和分发镜像，支持增量同步，方便团队协作。其底层整合 Linux 的 namespace、cgroups 和 UnionFS 技术，实现隔离、资源控制和高效存储，让应用“一次构建，到处运行”。

3.2 实验步骤

通过使用 `docker pull`，拉取了一个镜像名为 `ubuntu:16:04`

```
ubuntu@ubuntu-virtual-machine:~$ sudo docker pull ubuntu:16.04
[sudo] ubuntu 的密码:
16.04: Pulling from library/ubuntu
58690f9b18fc: Pull complete
b51569e7c507: Pull complete
da8ef40b9eca: Pull complete
fb15d46c38dc: Pull complete
Digest: sha256:1f1a2d56de1d604801a9671f301190704c25d604a416f59e03c04f5c6ffee0d6
Status: Downloaded newer image for ubuntu:16.04
docker.io/library/ubuntu:16.04
```

以 ubuntu 镜像为基础，启动 bash 并进行交互操作。在这里，我们执行了 cat/etc/os-release 命令

```
ubuntu@ubuntu-virtual-machine:~$ sudo docker run -it --rm \
> ubuntu:16.04 \
> bash
[sudo] ubuntu 的密码:
root@8f70b7982253:/# cat /etc/os-release
NAME="Ubuntu"
VERSION="16.04.7 LTS (Xenial Xerus)"
ID=ubuntu
ID_LIKE=debian
PRETTY_NAME="Ubuntu 16.04.7 LTS"
VERSION_ID="16.04"
HOME_URL="http://www.ubuntu.com/"
SUPPORT_URL="http://help.ubuntu.com/"
BUG_REPORT_URL="http://bugs.launchpad.net/ubuntu/"
```

拉取最新版本的 Nginx 镜像

```
ubuntu@ubuntu-virtual-machine:~$ sudo docker pull nginx:latest
latest: Pulling from library/nginx
3da95a905ed5: Pull complete
6c8e51cf0087: Pull complete
9bbbd7ee45b7: Pull complete
48670a58a68f: Pull complete
ce7132063a56: Pull complete
23e05839d684: Pull complete
ee95256df030: Pull complete
Digest: sha256:93230cd54060f497430c7a120e2347894846a81b6a5dd2110f7362c5423b4abc
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
```

通过 docker image 命令查看本地已有的镜像列表

```
ubuntu@ubuntu-virtual-machine:~$ sudo docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
nginx	latest	9592f5595f2b	7 days ago	192MB
ubuntu	22.04	1b668a2d748d	12 days ago	77.9MB
hello-world	latest	74cc54e27dc4	5 months ago	10.1kB
ubuntu	16.04	b6f507652425	3 years ago	135MB

```
FROM nginx
RUN echo '<h1>Hello, Docker!</h1>' > /usr/share/nginx/html/index.html
```

创建一个 ubuntu 容器

```
ubuntu@ubuntu-virtual-machine:~/mynginx/mynginx/mynginx$ sudo docker run ubuntu:14.04 /bin/echo 'Hello world'
Unable to find image 'ubuntu:14.04' locally
14.04: Pulling from library/ubuntu
2e6e20c8e2e6: Pull complete
0551a797c01d: Pull complete
512123a864da: Pull complete
Digest: sha256:64483f3496c1373bfd55348e88694d1c4d0c9b660dee6bfe5e12f43b9933b30
Status: Downloaded newer image for ubuntu:14.04
Hello world
```

使用 docker ps -a 查看所有容器

```
ubuntu@ubuntu-virtual-machine:~/mynginx/mynginx/mynginx$ sudo docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
02a0d40e04bf	ubuntu:14.04	"/bin/echo 'Hello wo..."	3 minutes ago	Exited (0) 3 minutes ago		brave_galileo
a50376cbe234	hello-world	"/hello"	33 minutes ago	Exited (0) 33 minutes ago		cool_mirzakhani

使用 docker stop 和 docker start

```
ubuntu@ubuntu-virtual-machine:~/mynginx/mynginx/mynginx$ sudo docker stop 02a0d40e04bf
02a0d40e04bf
ubuntu@ubuntu-virtual-machine:~/mynginx/mynginx/mynginx$ sudo docker start 02a0d40e04bf
02a0d40e04bf
ubuntu@ubuntu-virtual-machine:~/mynginx/mynginx/mynginx$
```

使用 docker exec-it 进入容器的交互式终端，并在容器中执行查找命令

```
ubuntu@ubuntu-virtual-machine:~/mynginx/mynginx/mynginx$ sudo docker exec-it 02a0d40e04bf /bin/bash
docker: unknown command: docker exec-it

Run 'docker --help' for more information
ubuntu@ubuntu-virtual-machine:~/mynginx/mynginx/mynginx$ ls
Dockerfile  'Dockerfile^C'
```

使用 docker volume creat 创建数据卷

```
ubuntu@ubuntu-virtual-machine:~/mynginx/mynginx/mynginx$ sudo docker volume creat my-vol
docker: unknown command: docker volume creat

Usage:  docker volume COMMAND

Run 'docker volume --help' for more information
```

3.3 实验结果与分析

Docker 实验围绕镜像，容器，仓库核心概念展开。从 Docker Hub 拉镜像，如 `docker pull nginx`，用 `docker images` 看本地镜像，了解镜像信息。编写 Dockerfile 定义基础环境，安装软件。容器操作方法的过程：`docker run` 启动容器，`docker ps` 看运行的容器，`docker stop`、`docker start` 控制启停，进容器终端能执行命令。网络方面，默认的 `bridge` 模式能让容器连外网，还能自己建网络，让容器互相通信。数据管理用数据卷和绑定挂载，存数据不怕容器删了丢失。Docker Compose 更方便，写一个 `docker-compose.yml`，一键启动多个服务，部署应用。Docker 把环境打包，让应用到处跑都一样，适合快速部署了。

第 4 章 Wireshark

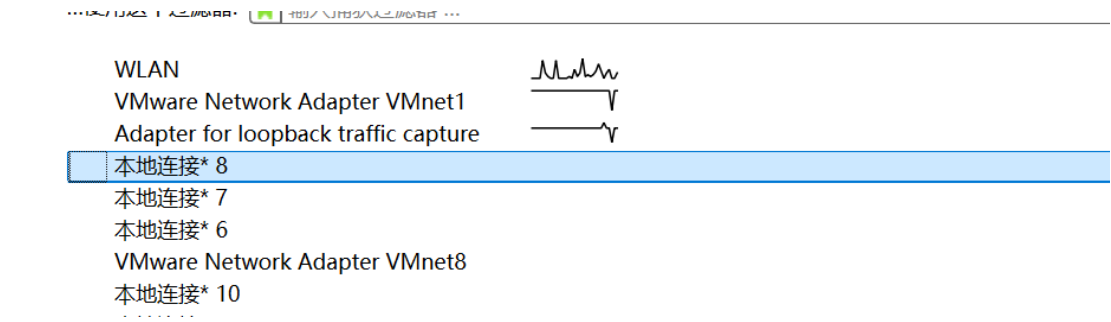
4.1 基本原理

Wireshark 是一款网络数据包分析工具，核心靠数据包捕获、解析与展示三大功能实现网络监控。它通过网卡进入 “混杂模式” 捕获流经的网络数据包，包括目标地址并非本机的流量。捕获后，借助内置的协议解析库，按 TCP/IP 等协议规范拆解数据包结构，提取源地址、目的地址、端口、数据内容等信息。解析后的内容以列表、详情和十六进制等多视图展示，支持按协议、地址等筛选，方便定位问题。其底层依赖 libpcap 等抓包库与系统接口交互，实现跨平台的数据包捕获与分析，助力排查网络故障、分析协议交互。

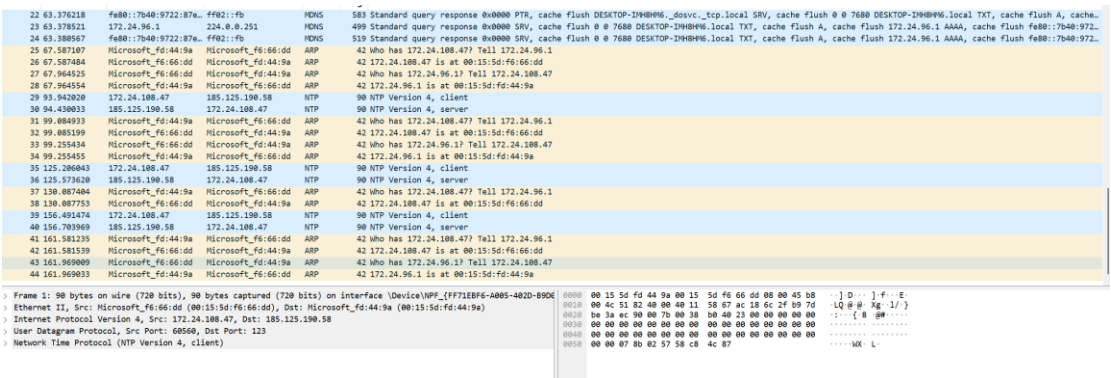
4.2 实验步骤

(1) 基本操作

在启动的 Wireshark 操作界面上直接双击要捕获的数据包网络接口



使用 start 按钮开始捕获，通过 stop 按钮停止捕获



使用 ip.src_host 过滤掉了源 IP 地址为 172.24.108.47 的数据包

ip.src_host == 172.24.108.47						
No.	Time	Source	Destination	Protocol	Length	Info
3	26.342229	172.24.108.47	185.125.190.58	NTP	90	NTP Version 4, client
9	57.647407	172.24.108.47	185.125.190.58	NTP	90	NTP Version 4, client
25	88.967703	172.24.108.47	185.125.190.58	NTP	90	NTP Version 4, client
31	120.315558	172.24.108.47	185.125.190.58	NTP	90	NTP Version 4, client
35	151.608420	172.24.108.47	185.125.190.58	NTP	90	NTP Version 4, client
41	182.915330	172.24.108.47	185.125.190.58	NTP	90	NTP Version 4, client
57	214.239936	172.24.108.47	185.125.190.58	NTP	90	NTP Version 4, client
63	245.593455	172.24.108.47	185.125.190.58	NTP	90	NTP Version 4, client
67	276.899765	172.24.108.47	185.125.190.58	NTP	90	NTP Version 4, client
73	308.212808	172.24.108.47	185.125.190.58	NTP	90	NTP Version 4, client

使用 tcp.port 过滤 tcp 端口为 5241 的数据包

tcp.port == 52415						
No.	Time	Source	Destination	Protocol	Length	Info
5	0.200620	127.0.0.1	127.0.0.1	TCP	45	52415 → 50968 [ACK] Seq=1 Ack=1 Win=251 Len=1
6	0.200670	127.0.0.1	127.0.0.1	TCP	56	50968 → 52415 [ACK] Seq=1 Ack=2 Win=255 Len=0 SLE=1 SRE=2

为了过滤出两种不同的数据包，可以使用 Ip.src_host==or tcp.port==过滤掉源 IP 地址为 127.0.01 的数据包或过滤 tcp 端口为 52415 的包

ip.src_host == 127.0.01 or tcp.port == 52415						
No.	Time	Source	Destination	Protocol	Length	Info
5	0.200620	127.0.0.1	127.0.0.1	TCP	45	52415 → 50968 [ACK] Seq=1 Ack=1 Win=251 Len=1
6	0.200670	127.0.0.1	127.0.0.1	TCP	56	50968 → 52415 [ACK] Seq=1 Ack=2 Win=255 Len=0 SLE=1 SRE=2

(2) 数据包分析方面

使用 ping 生成网络流量

```
C:\Users\Wandering Cloud>ping baidu.com

正在 Ping baidu.com [182.61.244.181] 具有 32 字节的数据:
来自 182.61.244.181 的回复: 字节=32 时间=49ms TTL=50
来自 182.61.244.181 的回复: 字节=32 时间=53ms TTL=50
来自 182.61.244.181 的回复: 字节=32 时间=57ms TTL=50
来自 182.61.244.181 的回复: 字节=32 时间=53ms TTL=50

182.61.244.181 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
    往返行程的估计时间(以毫秒为单位):
        最短 = 49ms, 最长 = 57ms, 平均 = 53ms
```

查看 ping 产生的 ICMP 包

icmp						
No.	Time	Source	Destination	Protocol	Length	Info
42	12.314395	192.168.109.66	182.61.244.181	ICMP	74	Echo (ping) request id=0x0001, seq=290/8705, ttl=128 (reply in 44)
44	12.363597	182.61.244.181	192.168.109.66	ICMP	74	Echo (ping) reply id=0x0001, seq=290/8705, ttl=50 (request in 42)
45	13.324552	192.168.109.66	182.61.244.181	ICMP	74	Echo (ping) request id=0x0001, seq=291/8961, ttl=128 (reply in 47)
47	13.378162	182.61.244.181	192.168.109.66	ICMP	74	Echo (ping) reply id=0x0001, seq=291/8961, ttl=50 (request in 45)
74	14.335735	192.168.109.66	182.61.244.181	ICMP	74	Echo (ping) request id=0x0001, seq=292/9217, ttl=128 (reply in 76)
76	14.392635	182.61.244.181	192.168.109.66	ICMP	74	Echo (ping) reply id=0x0001, seq=292/9217, ttl=50 (request in 74)
83	15.361373	192.168.109.66	182.61.244.181	ICMP	74	Echo (ping) request id=0x0001, seq=293/9473, ttl=128 (reply in 84)
84	15.415018	182.61.244.181	192.168.109.66	ICMP	74	Echo (ping) reply id=0x0001, seq=293/9473, ttl=50 (request in 83)

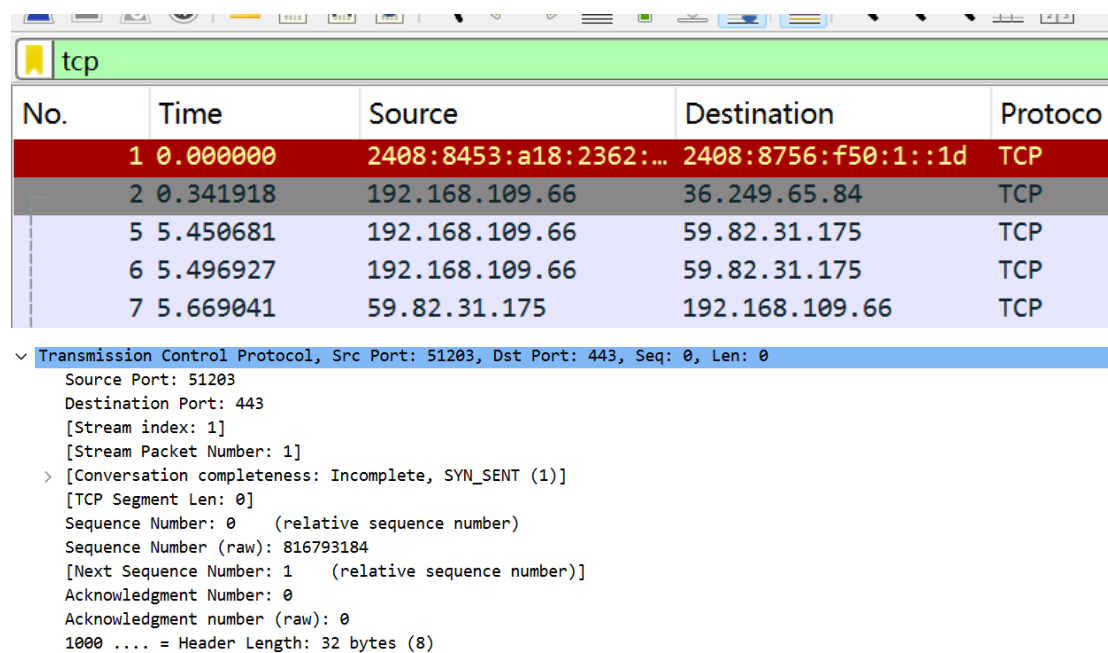
在 IP 包中查看地址，其中源 IP 地址是 192.168.109.66，目标 IP 地址是 182.61.244.181，TTL 值是 128 和服务类型是 1

```

...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 128
Protocol: ICMP (1)
Header Checksum: 0x0000 [validation disabled]
[Header checksum status: Unverified]
Source Address: 192.168.109.66
Destination Address: 182.61.244.181

```

过滤出 TCP 包，可以查看端口号，序列号，标志位

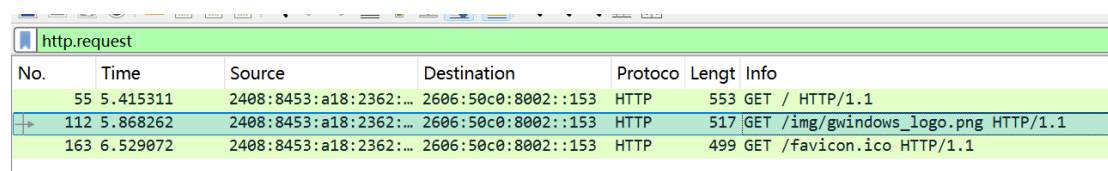


No.	Time	Source	Destination	Protocol
1	0.000000	2408:8453:a18:2362:...	2408:8756:f50:1::1d	TCP
2	0.341918	192.168.109.66	36.249.65.84	TCP
5	5.450681	192.168.109.66	59.82.31.175	TCP
6	5.496927	192.168.109.66	59.82.31.175	TCP
7	5.669041	59.82.31.175	192.168.109.66	TCP

Transmission Control Protocol, Src Port: 51203, Dst Port: 443, Seq: 0, Len: 0

- Source Port: 51203
- Destination Port: 443
- [Stream index: 1]
- [Stream Packet Number: 1]
- [Conversation completeness: Incomplete, SYN_SENT (1)]
- [TCP Segment Len: 0]
- Sequence Number: 0 (relative sequence number)
- Sequence Number (raw): 816793184
- [Next Sequence Number: 1 (relative sequence number)]
- Acknowledgment Number: 0
- Acknowledgment number (raw): 0
- 1000 = Header Length: 32 bytes (8)

在 HTTP 请求查看请求方法，其中 GET 就是请求方法



No.	Time	Source	Destination	Protocol	Length	Info
55	5.415311	2408:8453:a18:2362:...	2606:50c0:8002::153	HTTP	553	GET / HTTP/1.1
112	5.868262	2408:8453:a18:2362:...	2606:50c0:8002::153	HTTP	517	GET /img/gwindows_logo.png HTTP/1.1
163	6.529072	2408:8453:a18:2362:...	2606:50c0:8002::153	HTTP	499	GET /favicon.ico HTTP/1.1

(3) 协议分析

在 TCP 协议中，了解到了三次握手，我的电脑 IP 地址 192.168.2.199 与云服务器 43.139.52.212 进行了三次握手

Current filter: ip.addr == 43.139.52.212						
No.	Time	Source	Destination	Protocol	Length	Info
153	4.905485	192.168.2.199	43.139.52.212	TCP	66	65215 → 22 [SYN] Seq=0 Win=64240 Len=0 M
154	4.924355	43.139.52.212	192.168.2.199	TCP	66	22 → 65215 [SYN, ACK] Seq=0 Ack=1 Win=29
155	4.924458	192.168.2.199	43.139.52.212	TCP	54	65215 → 22 [ACK] Seq=1 Ack=1 Win=263424
156	4.950602	43.139.52.212	192.168.2.199	SSHv2	75	Server: Protocol (SSH-2.0-OpenSSH_7.4)
158	4.998222	192.168.2.199	43.139.52.212	TCP	54	65215 → 22 [ACK] Seq=1 Ack=22 Win=263168
159	5.029631	192.168.2.199	43.139.52.212	SSHv2	103	Client: Protocol (SSH-2.0-nsssh2_5.0.003
160	5.030017	192.168.2.199	43.139.52.212	SSHv2	1430	Client: Key Exchange Init
161	5.048089	43.139.52.212	192.168.2.199	TCP	60	22 → 65215 [ACK] Seq=22 Ack=50 Win=29312
162	5.048607	43.139.52.212	192.168.2.199	TCP	60	22 → 65215 [ACK] Seq=22 Ack=1426 Win=320
163	5.049395	43.139.52.212	192.168.2.199	SSHv2	862	Server: Key Exchange Init
164	5.050639	192.168.2.199	43.139.52.212	SSHv2	134	Client: Elliptic Curve Diffie-Hellman Ke
166	5.070374	43.139.52.212	192.168.2.199	SSHv2	710	Server: Elliptic Curve Diffie-Hellman Ke
167	5.073767	192.168.2.199	43.139.52.212	SSHv2	70	Client: New Keys

HTTP 请求: GET / HTTP/1.1 是, HTTP 响应: HTTP/1.1 200 OK

http						
No.	Time	Source	Destination	Protocol	Length	Info
113	4.564382	61.241.55.63	192.168.109.66	HTTP	274	HTTP/1.1 200 OK
193	13.776062	112.90.80.96	192.168.109.66	HTTP	447	HTTP/1.1 200 OK
499	29.281918	2408:8453:a18:2362::...	2600:1406:bc00:53::...	HTTP	547	GET / HTTP/1.1
507	29.598656	2600:1406:bc00:53::...	2408:8453:a18:2362::...	HTTP	1051	HTTP/1.1 200 OK (text/html)
508	29.633425	2408:8453:a18:2362::...	2600:1406:bc00:53::...	HTTP	487	GET /favicon.ico HTTP/1.1
513	29.930576	2600:1406:bc00:53::...	2408:8453:a18:2362::...	HTTP	422	HTTP/1.1 404 Not Found (text/html)
1306	47.185074	192.168.109.66	112.90.80.96	HTTP	182	Continuation
1308	47.262610	112.90.80.96	192.168.109.66	HTTP	218	HTTP/1.1 200 OK
3398	64.897976	2600:1406:bc00:53::...	2408:8453:a18:2362::...	HTTP	584	HTTP/1.0 408 Request Time-out (text/html)
3402	65.091614	2600:1406:bc00:53::...	2408:8453:a18:2362::...	HTTP	584	HTTP/1.0 408 Request Time-out (text/html)

DSN 请求包和响应包

```
Microsoft Windows [版本 10.0.26100.4484]
(c) Microsoft Corporation。保留所有权利。

C:\Users\Wandering Cloud>nslookup baidu.com
服务器:   Unknown
Address:  192.168.109.74

非权威应答:
名称:     baidu.com
Addresses: 182.61.244.181
          182.61.201.211

C:\Users\Wandering Cloud>
```

9	9.107905	192.168.109.74	192.168.109.66	DNS	142	Standard query response 0x0001 No such name PTR 74.109.168.192.in-addr.arpa SOA 168.192.IN-ADDR.ARPA
10	9.109710	192.168.109.66	192.168.109.74	DNS	69	Standard query 0x0002 A baidu.com
11	9.132665	192.168.109.74	192.168.109.66	DNS	101	Standard query response 0x0002 A baidu.com A 182.61.244.181 A 182.61.201.211

4.3 实验结果与分析

在 Wireshark 实操中, 我会启动软件选网卡抓包, 用过滤器筛特定 IP、端口的包。看数据包列表, 能懂编号、时间戳、源地址、协议这些字段, 点进去看详情, 从物理层到应用层, 逐层查信息, 像以太网帧的 MAC 地址、IP 包的地址、TCP

包的端口和标志位，HTTP 请求里的网址、请求头都能看到。分析协议时，能认出 TCP 三次握手、四次挥手，看 HTTP 响应状态码，HTTPS 虽加密但能看 SSL 握手，DNS 能查域名解析过程。抓包能帮我找网络问题，比如 TCP 重传影响速度，也能验证应用通信对不对。Wireshark 让我看透网络数据，以后排查问题更有办法了。

第 5 章 总结

通过本次通用工具操作实训，我系统学习了 Git、Docker、Wireshark 和 Burp Suite 等工具的核心功能与实际应用，掌握了从基础操作到高级分析的完整流程。在 Git 部分，我深入理解了工作目录、暂存区与本地仓库的交互机制，熟练掌握了分支管理、远程仓库同步及冲突解决等关键操作。通过实践，我认识到 Git 在版本控制和团队协作中的高效性，尤其是分支的灵活运用能够显著提升开发效率。Docker 实验让我熟悉了镜像与容器的生命周期管理，包括镜像构建、容器部署、网络配置和数据持久化。通过 Docker Compose，我体验了多容器应用的便捷编排，理解了容器化技术对开发环境一致性和服务隔离的重要性。网络分析工具 Wireshark 的使用让我掌握了数据包捕获与协议分析的核心技能。我成功解析了 TCP 三次握手、HTTP 请求响应等典型通信过程，并学会了通过过滤器精准定位问题。Burp Suite 的实践则增强了我在 Web 安全测试中的实战能力，尤其是请求拦截与修改的灵活应用。本次实训不仅提升了我的工具操作能力，更让我认识到这些技术在现代化开发、运维和安全领域的核心价值。未来，我将进一步结合理论知识与实践需求，探索工具在复杂场景下的深度应用，为职业发展奠定坚实基础。