SAST 2023 Summer Training
CTF | A Warmup for Linux & Git

jkjkmxmx @ yhx-12243

July 9, 2023

1 Linux

1.1 Introduction

1.1.1 什么是 Linux[4]

Linux 是一种自由和开放源码的类 UNIX 操作系统，其内核由 Linus Torvalds 在 1991 年发布。

Linux 也是自由软件和开放源代码软件发展中最著名的例子，只要遵循 GNU，任何个人和机构都可以自由地使用 Linux 的所有底层源代码，这使得它得到来自全世界软件爱好者和组织的开发支持。

Linux 以各种形式被广泛应用在各个领域，包括但不限于：

* 服务器、主机、超级计算机；
* 嵌入式系统（机顶盒、移动设备等）；
* 基础设施（红绿灯、工业传感器）。

1.1.2 发行版

我们平时使用的 “Linux” 严格来说是 Linux 发行版本，而 Linux 狭义上单指操作系统的内核。

发行版本在内核的基础上还包括安装工具、系统配置、图形桌面界面、各种 GNU 软件等，使得这个系统能够适用于各种使用目的。

常见的 Linux 发行版有 Debian、Ubuntu、Fedora、CentOS、Arch Linux 等。不同的发行版使用的软件不同，有些发行版本是设计成专门的目的（比如 Kali 用于网安）。[4]

以 Ubuntu 为例，Ubuntu 在每个偶数年的 4 月 (每两年) 发布一个长期支持版 (Long-Term Support)，每 6 个月发布一个版本。

我们提供的服务器安装的发行版是 Ubuntu 22.04 (LTS)，代号 Jammy Jellyfish。

1.1.3 接触 Linux 的方式

我们接触 Linux 的方式通常有以下几种：[4]

* 安装 Linux 系统 (单系统、Windows & Linux 双系统)；
* WSL (英语：Windows Subsystem for Linux)；
* 虚拟机 (Virtualbox、Docker)；
* 服务器远程连接 (SSH、RDP)；

有条件的同学可以给电脑装一个 Linux 系统 (或双系统)，~~这样你就能在各种配置中度过一段漫长而有趣的时光子~~。

对于其他同学，Windows 的话推荐使用 WSL 2 (毕竟是 Type-1 的虚拟机，有性能支持)，~~如果只想要一个终端的话可以用 Git Bash~~；对 macOS 的话原生的终端就有比较接近的体验 (推荐使用 iTerm 2)，或者使用 Docker。当然有服务器的同学可以直接用各种服务器 (现代的服务器基本都是 Linux)。

1.2 课前准备

参考链接：

* https://learn.microsoft.com/zh-cn/windows/wsl/install
* https://blog.csdn.net/yushuzhen2008/article/details/104944579

1.3 GUI, CLI, Shell 和 Terminal

见 https://docs.net9.org/basic/linux/#basic-concepts。

1.4 学习资料

网上的 Linux 学习资料非常多，同学们需要善于寻找资源：

* https://docs.net9.org/basic/linux/ ← **务必熟读，其中的内容本文不再重复出现**
* https://www.runoob.com/linux/linux-tutorial.html
* https://ryanstutorials.net/linuxtutorial/
* https://missing.csail.mit.edu/

当遇到问题时，要善于利用 Bing、Google 等搜索引擎，以及 Stack Overflow、Stack Exchange 等。当然在 2023 年的今天，ChatGPT/GPT-4 等工具也一直是你的好伙伴 (对于比较基础的问题效果还是很不错的)。

当然，在遇到不会的命令时，还要养成看文档的习惯，这一部分可以参考 1.6 节。

1.5 Shell

Shell 是终端编程中常见的 “语言”，Shell 变成为典型的命令式编程 (区别于声明性编程)，常用于写快餐式的脚本以及充当 “胶水” 的作用，~~和 Python 比较类似~~。

1.5.1 Shell 方言

Shell 并不是一种固定的语言，它是衍生出的许多方言 (扩展) 的统称，如 `sh` (原生), `bash`, `zsh`, `dash`, ~~`fish`, `csh`, `ksh`~~ 等。

比较常见的是 Bash (Ubuntu 默认) 和 Z shell (zsh, macOS 默认)。当然这些 shell 自身也会有相对应的扩展，比如服务器上默认安装的 Z shell 扩展 “Oh My Zsh”。

通过 `type` 命令[1]可见形如 `cd`, `if`, `pwd`, `alias`, `setopt`/`shopt` 等命令返回了 “shell 内建”/“shell 关键字”，它们统称**内部命令**；而形如 `ls`, `wc`, `ps`, `cp`, `find`, `mkdir`, `sudo` 等命令会返回一个对应文件所在的路径，它们统称**外部命令**[2]。

* **内部命令**是嵌入在各个 Shell 中的 (写在对应的源码里)，~~就像你们写 Shell 大作业时的一堆 if/else 里讨论的命令~~，不创建新进程，执行速度较快，缺点是不同的 Shell 的实现可能会不一样，就是说方言可能会比较多。
* **外部命令**是 Linux 系统中独立存在的外部程序，由于是外部程序，因此在所有 Shell 看来表现几乎一样，调用时会创建新的子进程。

  当输入文件名为一个标识符时，Shell 通常不会解析为相对路径，而是解析为命令，然后按照内部优先，外部其次的顺序进行查找。其中外部命令的查找会根据 `PATH` 环境变量按顺序依次查找。

  具体的优先级可以使用 `type -a <command>` 来查看，以得知有哪些结果被覆盖。

[1] 有些教程用建议使用 `which` 命令，但是在有一些 Shell (Sh 和 Bash) 中 `which` 是**外部命令**，因此无法查阅如 `if` 等**内部命令**。而 `type` 在常见的 Sh, Bash, Z shell 中均为内部命令，可以查询到所有命令的信息。

[2] 实际上还有别名，函数等形式，可以使用 `type -w` (Bash 中为 `type -t`) 来查看类别。

1.6 在 Linux 中查文档

在 Linux 以及之后的各种工作中，查文档是必不可少的技能。除上网查询外，Shell 中查文档的方法主要有如下几种：

* `<command> --help`，**只能用于 (文档良好的) 外部命令** (大多数外部命令都有良好的文档)。
* `help <command>`，**Bash 中查询内部命令的方式**。
* `man <command>`，比较通用，**非常牛逼，非常详细，而且通常由官方撰写的命令手册**，强烈推荐使用！

  `man man` 可以查询 `man` 的使用方式，~~没想到吧，这就叫元学习！~~

  所有这些手册 (manual) 分为 9 类 [2]：

  1. 可执行程序或 shell 命令；
  2. 系统调用 (内核提供的函数)；
  3. 库调用 (程序库中的函数)；
  4. 特殊文件 (通常位于 `/dev`)；
  5. 文件格式和规范，如 `/etc/passwd`；
  6. 游戏；
  7. 杂项 (包括宏包和规范)，如 `man(7)`, `groff(7)`, `man-pages(7)` 等；
  8. 系统管理命令 (通常只针对 root 用户)；
  9. 内核例程 (非标准)。

  通过 `man <类别> <命令>` 可以具体指定在哪一类中查。大家可以体会一下 `man 1 printf` 和 `man 3 printf` 的差别[3]。

  另外一个常用的选项是 `-k`，就当你不知道要对什么 `man` 时，查询所有能匹配正则表达式的手册列表。

  此外，进入手册后，Shell 通常会使用一个叫做 `less` 的 Pager (说人话就是你和使用 `less` 命令一样)，这是一个和 Vim 类似的环境。此时你可以 `man less` 或直接按下 “`h`” 键 (或直接搜索) 寻求帮助，比如常见的 “`/`” 表示搜索。

  有关 `man` 的更多用法详见 `man man`。
* `man bash/man builtins` (Bash) 和 `man zshbuiltins` (Z shell) 用于查询所有内部命令的文档。

  ~~相信大家也被 Z shell 里没有 `help` 命令而且 `man if` 跳出的东西云里雾里所困扰吧（逃~~
* `tldr <command>`。对于一些不常用的命令，你不打算花时间精雕细琢地学习，只想快速知道最常见的用法 (范式，模板)。这时，`tldr` 是一个非常不错的选择。`tldr` (Too long; didn't read) 提供了一些命令的非常常见的套路用法，以加快你的工作效率。

注：在这之后提到如果有没出现的 “前置技能”，就表示建议使用 `man` 或 `tldr` 工具进行相关自学。

再次提醒，学会主动查手册是一项非常非常非常重要的技能，之后的命令介绍会非常简略，希望大家学会自己查 `tldr` 获取主要用法，查 `man` 获取手册!!!

再次提醒，学会主动查手册是一项非常非常非常重要的技能，之后的命令介绍会非常简略，希望大家学会自己查 `tldr` 获取主要用法，查 `man` 获取手册!!!

再次提醒，学会主动查手册是一项非常非常非常重要的技能，之后的命令介绍会非常简略，希望大家学会自己查 `tldr` 获取主要用法，查 `man` 获取手册!!!

1.7 SSH 相关

[*主要命令：ssh, scp, ssh-keygen, ssh-copy-id*]

命令的用法请使用 `man` 以及 https://docs.net9.org/basic/linux/#ssh 进行学习。值得注意的一点是，可以在家目录下配置 `.ssh/config` 文件来将记住常用的服务器配置 (VSCode 也会使用这个！) 详细的规范见 `man ssh_config`。

1.8 Shell 语法

[方便起见，本节内容偏向 *Z shell*。]

见表 1，扩展自 [5]。官方文档可参考 https://zsh.sourceforge.io/Doc/Release/index.html。

命令	例子	介绍	手册
`# comment`	`# this is a comment`	注释	`man zshmisc`
`<VARIABLE>=<VALUE>`	`ANSWER=42`	定义一个变量，**注意等号两边不能有空格，且 Shell 大小写敏感**	`man zshparam`
`${VARIABLE}`	`echo ${ANSWER}`	使用变量，无歧义时可省略大括号	`man zshparam`
`$NUM`	`echo $1`	使用参数，如 `./script arg1 arg2 arg3` 中 `$2` 即为 `arg2`	`man zshparam`
`$@` 或 `$*`	`echo $*`	所有参数的数组	`man zshparam`[4]
`expr`	`expr 2 + 4`	计算数学表达式	`man expr`
`` `COMMAND` `` 或 `$(COMMAND)`	`echo $(expr 2 + 4)`	取命令结果	`man zshexpn`
`{A,B,C}`	`cat file{A,B,C}.txt`	按照分配律展开	`man zshexpn`
`{low..high}`	`touch test{1..10}.in`	范围展开	`man zshexpn`
`~`	`cd ~`	展开你的家目录	`man zshexpn`
`~USER`	`echo ~root`	展开某个用户的家目录	`man zshexpn`
`?, *, **, [...]`	`rm **/A?[1-5].log`	文件名匹配 (Glob) 语法，**使用引号可以避免自动展开** (`zshmisc`)	`man zshexpn`

表 1: 常用语法

其它复杂的语法 (如 `() (()) [] [[]] {} if for case select` 数组、算术运算定义函数等) 可见 `man zshexpn`，`man zshparam`，`man zshmisc` 等。[5]

所有文档可通过 `ls /usr/share/man/man1/zsh*` 或 `man zshall` 查询，当然也可查询 https://zsh.sourceforge.io/Doc/Release/index.html (其实内容是一样的说) 或网上相关教程 (不一定靠谱)。

1.8.1 环境变量

环境变量是一类特殊的变量，有点像局部变量，它的一大特性是**可以继承给它的子进程**，从而广泛应用于外部命令们的参数配置。

环境变量的定义有如下几种方式：

* `export <VARIABLE>=<VALUE>`：定义环境变量。
* `<VARIABLE>=<VALUE> <command>`：在这一条命令中拥有该环境变量，命令结束后变量消失 (`man zshmisc`)。
* 使用其它程序设置。

环境变量可以用程序或 `unset <VARIABLE>` 清除。

* `declare` / `typeset` 可用于查看所有定义的变量。
* `env` 可用于查看所有定义的环境变量，以及运行相关程序。

1.8.2 调用其它脚本 [5]

在 Shell 中调用其它脚本有两种方式：`source` (简写 `.`) 或子 Shell (`zsh <script>` 或使用 Shebang)，当然还有一种方法是通过 `exec` 直接替换当前进程，由于这个区别比较大就不讨论了。

[4] 不同 Shell 对这种变量的处理方式大相径庭。Bash 的处理比 Z shell 更加混沌邪恶。在 Z shell 中 `$*` = `$argv`，以及对应的 `$ARGC`，具体区别可见 `man zshparam`。

[5] 注：如果你真的想在 Shell 里使用复杂的语法，建议使用 Python 等方便的语言。

主要区别是：source 方法在本 shell 执行，子 Shell 方法会新建子终端执行。
脚本中的局部变量定义仅属于执行它的终端。

```
echo 'ANSWER=42' > script.sh
zsh script.sh # 变量在子终端定义
echo ${ANSWER} # 对本终端，变量不可见
source script.sh # 变量在本终端定义，也可用 . script.sh
echo ${ANSWER} # 变量可见
```

1.9 [6] Shell 选项

[方便起见，本节内容偏向 *Z shell*。]
Shell 选项，有些地方有称之为 Shell 专有变量。
对 Z shell 而言，手册可在 man zshoptions 中查看，使用 setopt 和 unsetopt 设置。
表 2 列出了一些常用的 Z shell 选项。

选项	启用 (setopt) 效果	禁用 (unsetopt) 效果
autocd	对于无法解析的命令，看成前面有 cd。如 .. 默认成为 cd ..	无前述效果
interactivecomments	在交互式终端中允许使用 # 注释	交互式终端不能注释，脚本中仍可注释
glob	启用 Glob (man zshexpn)	禁用 Glob
nomatch	对于无法匹配的 Glob 报错	对于无法匹配的 Glob，直接原样返回 (建议开启这个，因为有些命令需要原样的 * 等字符，避免频繁打引号)
globdots	Glob 中 * 匹配隐藏文件	Glob 中 * 不匹配隐藏文件
globstarshort	使用 ** 代替 **/* 递归列举文件[7]	单纯的 ** 等价于 *
nullglob	对于无法匹配的 Glob，返回空串而不报错会原样返回	遵循 nomatch
extendedglob	允许使用更高级的语法 (具体参见 man zshexpn)	禁用高级语法

表 2: 常用的 Z shell 选项

* 使用不带参数 setopt / unsetopt / set -o / set +o 可以查看所有 Shell 选项。
* 使用 [[-o <option>]] 可以检查某个 Shell 选项是否设置 (man zshmisc)。

1.10 一些命令

这里还有一些~~可能在 Puzzle 中用得到的~~命令，具体方法请自行使用 man 和 tldr (记住，他们真的是很好的老师，我已经没什么好写的了)。

1.10.1 进程

* ps / top / htop：查询进程信息
* uptime：查询系统启动时间，用户数以及平均负载 ~~(其实就是 top 的第一行)~~
* exec：用接下来的程序替换当前进程
* kill / killall：杀死进程
* ... & / fg / bg / jobs / nohup：前后台切换，~~避免《人类的群星闪耀时》~~(查不到怎么用？看看这里！)

[6] 为致敬 Knuth 著的 TEXbook 的符号，表示可以选读的进阶内容。
[7] 递归列举的功能在 Bash 中需要通过打开 globstar 选项来实现。

1.10.2 文件系统

* `cat`：输出并连接文件
* `tee`：将输入复制一份 (标准输出和文件)
* ~~`tac` / `rev`：花里胡哨地输出文件~~
* `stat`：显示文件信息
* `wc`：统计行数等信息
* `sort`：排序
* ~~`tsort`：拓扑排序~~
* `uniq`：去重
* `grep`：搜索文件内容
* `awk` / `cut` / `sed`：简单文件处理
* `head` / `tail`：截取文件 (`tail -f` 比较有用，通常用于输出日志)
* `du`：显示已用空间
* `df`：显示可用空间
* `less`：友好地展示文件内容 (允许手动滚屏)
* `realpath`：显示经过解析的绝对路径 (所有符号链接均经过解析，从根目录开始)
* `basename` / `dirname`：分离目录名和文件名
*  `split`：分裂文件 (用于制作分卷压缩包)
* `dd`：拷贝文件的底层接口

1.10.3 网络 [5]

* `ifconfig` / `ip`：查询网络配置
* `netstat`：查询连接情况
* `ping`：检查网络是否连通
* `curl` / `wget`：命令行网络 (HTTP 等协议) 连接
*  `tcpdump`：抓网络包
* `nc`：手动发/听网络包
* `dig` / `nslookup`：DNS 解析

1.10.4 其它

* `clear` 清屏
* `history` 查询历史记录
* `ulimit` 修改 Shell 的资源限制
* `alias` / `unalias` 定义与取消定义别名
* `tar` / `gzip` / `gunzip` / `zip` / `unzip`：各种不同格式的压缩工具
* `shutdown` / `halt` / `reboot`：关机及重启
* `date`：显示或更改系统时间
* `uname`：查询系统相关信息 (`-a`)
* `lsb_release`：(Ubuntu) 查询版本信息 (`-a`)

* `service` / `systemctl` 操作系统服务
* `w` / `who`：查询登录用户
* ~~`write`：聊天（记住 Linux 是多用户操作系统）（可以在上某些课时恶搞让别人终端突然收到消息）~~
* ~~`wall`：广播消息~~
* ~~`factor`：分解质因数~~
* ~~`tty`：查询当前 tty~~

2 Git

2.1 Introduction

在代码开发过程中，总会遇到一些问题。代码不知道改了哪里不能运行了，想要回退怎么办？多个人同时想对代码进行修改（并行开发），开发完后怎么合并？显然，有一个能帮助我们解决这些问题的工具将极大的加快我们开发的速度。而这种工具，便是版本控制软件。

版本控制软件帮助使用者找出：

* 不同版本之间的差异；
* 谁做出了这个修改；
* 什么时候做出了这个修改；
* 做出修改的人给出的修改理由。

那么，什么是分布式的版本控制软件呢？为了了解清楚，我们需要区分中央版本控制系统 (CVCS) 和分布式版本控制系统 (DVCS) 的区别。

中央版本控制系统必须同时存在服务端和客户端。当进行代码备份时，客户端会向服务端发出请求，并将此次修改的内容发送到服务器当中去，服务端收到请求后，会将代码存储在服务器当中；同样当客户端想查看某一个版本的修改内容或者想恢复到某一个版本之时，客户端也会发送请求到服务端，服务在端接收到请求之后会做出相应的处理并返回给客户端。也就是说，在业务流程过程中，服务端必须存在，所有请求必须经过服务端的处理。

分布式版本控制器，主要是将备份的代码以及记录完全独立在本地存储。比如说当你想将代码恢复到某一个版本的时候，本地版本控制器不需要依赖网络便可以完成此操作，因为本地版本控制器拥有完整独立的一套控制系统。[3]

Git 作为世界上最流行的 DVCS，占了开源市场版本控制系统约 74% 的份额[8]，其余比较常见的有 svn (CVCS, 22%, 代表者 texlive 等)，Mercurial (DVCS, 1%, 代表者 nginx 等)。而且，很多 DVCS 或多或少受到 Git 的影响 (如 Mercurial/hg 等)，因此学习 Git 也会对之后学其他 DVCS 带来便利，这也是我们学 Git 的理由之一。

2.1.1 什么是 Git[3]

Git 是 Linux 之父 Linus Torvalds 为开发 Linux 内核而建立的一个分布式版本控制软件。

Git 除了版本控制软件本身的优势以外，还可以：

* 通过查看 Git history，开发者可以看到一个项目开发的时间线；
* 通过 Git branch (分支)，开发者可以在不用担心影响主代码的情况下进行开发；
* ……

[8] https://openhub.net/repositories/compare, 2023.7.

2.2 课前准备

Git 下载 (官网)：https://git-scm.com/downloads
Git-for-windows [7]：

* https://gitforwindows.org
* 国内镜像：https://npm.taobao.org/mirrors/git-for-windows/
* 清华镜像：https://mirrors.tuna.tsinghua.edu.cn/github-release/git-for-windows/git/

Linux (一般自带，以 Ubuntu 为例)：`sudo apt install git`
macOS (一般自带)：`brew install git` 或 https://sourceforge.net/projects/git-osx-installer/ [7]。

2.3 学习资料

网上的 Git 学习资料非常多，同学们需要善于寻找资源：

* https://docs.net9.org/basic/git/ ← **务必熟读，其中的内容本文不再重复出现**
* https://www.runoob.com/git/git-tutorial.html
* https://www.liaoxuefeng.com/wiki/8960434880296000
* https://git-scm.com/book/zh/v2
* https://learngitbranching.js.org/?locale=zh_CN ← 学习 Git 分支原理一定要玩的小游戏！
* https://github.com/b1f6c1c4/learn-git-the-super-hard-way ← 学习 Git **底层原理**的好教程！
* https://education.github.com/git-cheat-sheet-education.pdf ← 一些重要命令的 cheat-sheet

当遇到问题时，要善于利用 Bing、Google 等搜索引擎，以及 Stack Overflow、Stack Exchange 等。当然在 2023 年的今天，ChatGPT/GPT-4 等工具也一直是你的好伙伴 (对于比较基础的问题效果还是很不错的)。
当然，在遇到不会的命令时，还要养成看文档的习惯，这一部分可以参考 1.6 和 2.4 节。

2.4 如何查文档

除搜索引擎/ChatGPT 等方式外，在 Git 中查文档主要有如下方式：

* 官网：https://git-scm.com/docs，非常权威，可切换语言。
* 以下三种方式等价，且内容和之前官网本质相同：
  - `git <command> --help`
  - `git help <command>`
  - `man git-<command>`[9]

  为统一起见，下文均使用第二种规范。

  特别地，除了命令外，git 还提供了其他重要的文档，比如 `git help revisions` 给出了 Git 中指定一个版本或范围的所有方法的详细文档。
* `tldr git <command>`，见 1.6 节。

2.5 Git 配置

[直冲官方版：https://git-scm.com/docs/git-config/zh_HANS-CN 或 git help config]
Git 配置根据作用域分为好几类，主要的三类从小到大见表 3。
显然可知，用户名，邮箱~~, GPG key~~ 等配置是关于用户的配置，应该选择全局，所以配置时应当带上 `--global`。
Git 的配置里包含很多好玩东西，比如别名等，在后面会介绍。

[9]和 1.8 节类似，所有手册页可通过 `ls /usr/share/man/man*/git*` 查询 (Linux)。

作用域	英文名	生效范围	存储地址
局部	local	该仓库	./[10].git/config
全局	global	本用户	~/.gitconfig
系统	system	整台计算机	/etc/gitconfig (通常)

表 3: Git 配置的作用域

2.6 工作区、暂存区和版本库

补充：`git checkout` 自 Git 2.23 起不建议使用 (deprecated)，官方建议使用分工更明确地 `git switch` (切换分支，分离 HEAD) 和 `git restore` (恢复文件)。

* `git status`：查看当前状态，并给出提示 (**非常常用！很多时候 Git 的提示非常贴心，会告诉你相应状态和接下来应当做的操作**)；
* `git add`：工作区 (worktree) → 暂存区 (index/staged)
* `git commit`：暂存区 (index/staged) → 版本库 (HEAD)
* `git restore`：恢复文件
* `git diff`：显示版本间变化

更新后的图：

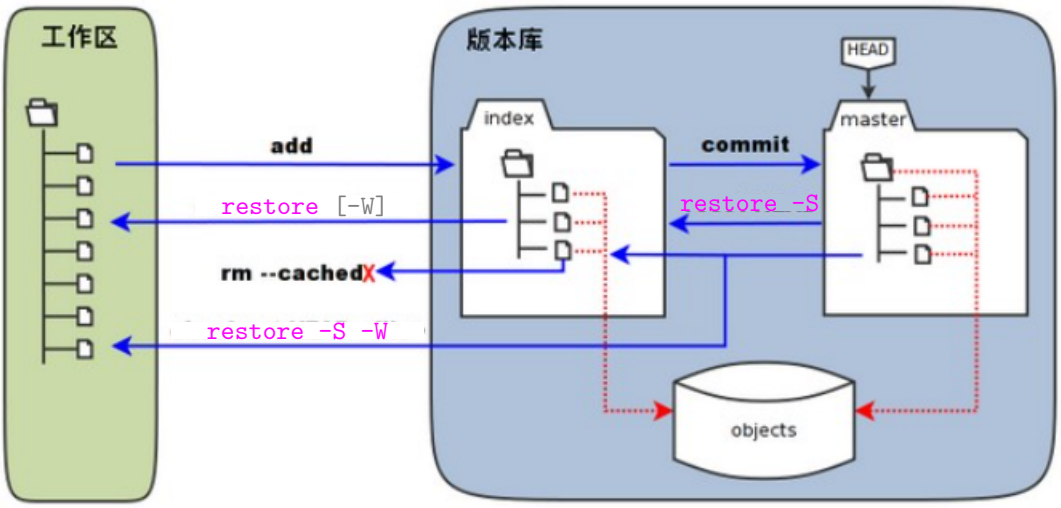

图 1: Git 工作区、暂存区和版本库

2.7 分支操作

强烈建议从 https://learngitbranching.js.org/?locale=zh_CN 学习！
强烈建议从 https://learngitbranching.js.org/?locale=zh_CN 学习！
强烈建议从 https://learngitbranching.js.org/?locale=zh_CN 学习！

* `git branch`：分支管理 (具体用法见 `git help branch`)
* `git switch`：切换分支 (比较常见的 `git switch -c`，创建并切换)
* `git tag`：标签管理 (https://www.liaoxuefeng.com/wiki/896043488029600/900788941487552)

[10]. 指仓库所在目录。

* `git log`：展示历史
* `git show`：显示提交内容[11]
* `git diff`：显示版本间变化
* `git merge`：冲突处理之合并 (见前述链接)
* `git rebase`：冲突处理之变基 (见前述链接，其中 `git rebase -i` 较常用，可用于合并多个提交，简化版本树)

2.8 远端操作

* `git remote`：远端管理
* `git clone`：克隆远端仓库
* `git fetch`：收取远端变化
* `git push`：推送变化到远端
* `git pull`：`git fetch` + `git merge` (或 `git rebase`)
* `git submodule`：子模块管理

分支跟踪：见 [6] 或 `git help branch`。

2.9 其他命令

这里还有一些~~可能在 Puzzle 中用得到的~~命令，具体方法请自行使用 `git help` (记住，他们真的是很好的老师，我已经没什么好写的了)。

* `git blame`：查看贡献者 ~~罪魁祸首之谁写的 bug~~
* `git shortlog`：统计不同用户的贡献 ~~找出摆烂的队友~~
* `git reflog`：查询引用日志，用于误操作恢复
* `git bisect`：在版本树上二分查找定位 bug
*  `git gc` / `git prune` / `git fsck`：用于系统检查和垃圾清理

2.10 Git 别名

[直冲官方版：*https://git-scm.com/book/zh/v2/Git-%E5%9F%BA%E7%A1%80-Git-%E5%88%AB%E5%90%8D*]

相信大家也可以发现，在平常使用 Git 时，经常会打出一长串又臭又长的命令，此时 Git 别名可以使使你的 Git 体验更简单、容易、熟悉。

Git 的别名实际上就是节为 `alias` (键为 `alias.*`) 的配置，因此可以使用 `git config` 直接配置。由于 Git 别名通常给个人使用，因此配置时最好带上 `--global`。

在下发的服务器的 `~/.gitconfig` 文件中就有一些被注释掉的别名配置 (这些是我使用的)，大家可以根据自己的需要定义别名。

2.11 gitignore

[直冲官方版：*https://git-scm.com/docs/gitignore 或 git help ignore*]

在平时的工作中，经常会出现一些垃圾文件，如 C++ 的 `.o`, `.exe`，LaTeX 的 `.log`, `.out`，Node.js 的 `node_modules`，~~以及 macOS 的万恶之源 `.DS_Store`~~。我们不希望这些 (可能很庞大) 的文件被 Git 跟踪 (或一不小心被加入暂存区甚至版本库)，我们需要通过 Git 提供的 ignore-list 规则来告诉 Git 不需要管理的文件。

简单地来说，你只需要在仓库的任意位置创建一个名为 `.gitignore` 的文件，然后一行写一个规则，使得满足这个规则的文件不被考虑。这样，你就可以放心使用 `git add .` 而不用担心加入奇奇怪怪的文件了。[6]

.gitignore 文件的具体语法见 https://git-scm.com/docs/gitignore#_pattern_format。

[11]冷知识：`git log` 和 `git show` 的底层实现是完全一致的，只是默认选项不同而已。

```
# 忽略 folder 下的全部内容，这包括根目录下的 /folder/，也包括某个子目录下的 /child/folder/ 内容：
folder/*
# 忽略根目录下的 /folder/ 下的全部内容：
/folder/*
# 忽略所有 .cpp 文件：
*.cpp
# （尽管忽略所有 .cpp 文件，但）不过滤 special.cpp 文件：
!special.cpp
```

扩展：.gitignore 虽然能 ignore 掉一些文件，但其本身还是会被 Git 跟踪。有些时候你~~可能因为独特的环境~~导致产生一些只有你的工作区才有的脏东西，那么你不希望将它们添加到 .gitignore 里。显然你不能把 .gitignore 自己给 ignore 掉。此时，对于这一类文件，可以在 `./.git/info/exclude` 里填写，语法和 .gitignore 相同。与 Git 配置类似，它也有全局 (用户) 版本：`~/.config/git/ignore`。

2.12 不通过 GitHub 直接交流

一个常见的需求是你在写项目的时候需要和给你的分配某个服务器同步，然后本地写代码 (编辑起来比 Remote-SSH 快) 在服务器上调试运行 (有特定环境，有算力等)。

此时一个常见的方法是通过 GitHub 去同步。但是这样就变成了中心化的方式，没有充分利用 Git 作为 DVCS 的优势。而 `rsync` 在这一方面又显得比较笨重，比如在增量更新，分支切换等方面。

其实，**当你可以 SSH 一个服务器的时候，你就可以把它当成一个 SSH 协议的 Git 服务器！**具体操作如下 (假设你 SSH 使用的命令为 `ssh user@hostname`，有端口的情况通过 `.ssh/config` 去掉)：

* 设**远端**的仓库地址为 `/path/to/repository`，建立一个空仓库或原仓库的副本 (第一次操作)；
* 在**远端**的局部 (仓库) 配置上 `receive.denyCurrentBranch` 值为 `updateInstead` (`git help config` / `git help hooks`)；
* 然后在你**本地**的仓库中配置 `git remote add <name> user@hostname:/path/to/repository/.git`；
* 这样就可以愉快地 `fetch`/`push` 啦！

2.13 其他技巧

Git 还有很多小技巧，限于篇幅原因这里不一一介绍，有兴趣的同学可以查阅相关资料自行学习。

* Git LFS：用于管理大文件，如模型、数据等。https://git-lfs.com/
* 提交签名：给你的 Git 提交签名，这样你的提交就会旁边就会有 Verified 字样，防止别人伪造。https://docs.github.com/zh/authentication/managing-commit-signature-verification
* GitHub Actions：CI/CD 的好帮手，减轻人类的负担。

zuò yè
3 实战

3.1 服务器连接

[前置技能：*ssh* 命令]

我们会按照报名问卷的填写情况为大家分配服务器账号，在获取到你的服务器用户名、IP 地址与账号后，请使用以下命令连接到服务器：

```
ssh train@59.66.131.240 -p <port>
```

所有人的初始账号均为 `train`，初始密码均为 `sast2023`。具体的端口号 (即 `<port>` 一栏) 会通过清华邮箱下发。

3.2 公钥登录 (推荐)

[前置技能：*ssh-keygen, ssh-copy-id* 命令]
在登录成功后请尽快配置公/私钥登录，并修改强密码或撤销密码登录，简要流程如下：

* (在本地) 直接输入命令 `ssh-keygen` (注：可选参数 `-t ed25519`，更多参数见命令手册)，会在家目录的 `.ssh` 文件夹下生成一对公私钥 (`id_rsa.pub` 和 `id_rsa`)。
* (在本地) 运行 `ssh-copy-id train@59.66.131.240 -p <port>` (格式和 3.1 节相同，更多参数见命令手册)，输入密码后即可完成 “免密登录” 配置。

3.2.1 撤销密码登录 (可选)

[前置技能：*sshd* 服务相关原理]
当你获得管理员 (root/sudo) 权限后，修改 `/etc/ssh/sshd_config` 文件：

* 将第 57 (约) 行的 `#PasswordAuthentication yes` 改为 `PasswordAuthentication no`;
* 运行 `sudo services ssh reload` 重新读取配置。

3.3 CTF — 简介

为保证趣味性，我们沿用之前几届的传统，采用 CTF (Capture the Flag) 的形式进行。你需要在服务器环境中寻找格式形如 `sast2023{*******}` 的字符串，其中 `*******` 可以包含任何非空白 ASCII 可见字符。我们将这种字符串称为 “flag”。每当你找到一个 “flag”。你就可以通过 3.4 节中介绍的提交方式进行提交。[1]

3.4 CTF — 提交方式

[前置技能：简单的命令行网络工具，如 *curl, wget* 等]
不提供额外的提交程序，请使用命令行发送网络请求的方式进行提交。推荐使用 `curl`。
提交的命令如下 (尖括号 `< >` **不**输入，花括号 `{ }` **要**输入)：

```
curl -d 'id=<你的学号>' -d 'flag=sast2023{*******}' http://59.66.131.240:50000/submit
```

对于如外校等没有学号的用户，可以直接去掉学号一栏，即使用如下命令提交。此时该命令仅用于检测正确性，不提供累积计分与排名功能。

```
curl -d 'flag=sast2023{*******}' http://59.66.131.240:50000/submit
```

输入该命令后，你的终端会返回你的提交是否有效。简单起见，如果返回以 `"Success":` 开头说明提交成功，否则会给出对应的错误信息。

此外，你可以在浏览器直接访问 http://59.66.131.240:50000/ 查看当前 “flag” 的获取情况以及排行榜。排行榜默认按照总分降序排列，同分按最后一次提交时间升序排列。你也可以点击表头更改排序方式。

3.5 CTF — 注意事项与 FAQ

* 所有的 “flag” 分为三大区：Linux, Git, Bonus，分别包含 9, 6, 6 个 “flag”，占总数的 58, 36, 6 分。
* 藏匿的 “flag” 会以多种形式出现，一般而言，出现格式正确的字符串就是 “flag”，除非有特殊说明，要求你寻找其中一个特定的 “flag” (这种现象在 Git 区中比较常见)。
* 欢迎与其他人交流讨论思路，以及向 ChatGPT/GPT-4 等工具求助。但请不要直接将 “flag” 告诉别人，这样会影响其他人的游戏体验。
* 提交 “flag” 的服务器和分配给你的服务器是同一个服务器，所以不要尝试做出伤敌一千自损八百的事情。
* 由于🕊️🕊️🕊️的原因没做复杂的身份验证，所以请只用自己的学号提交 ~~(真有人喜欢给别人送分吗？)~~。

* “mails” 文件夹用于获得服务器的管理员权限以进行进一步操作，该文件夹与 “flag” 无关，且解锁所有三个区的 “flag” 都无需用到该权限 (即使用 “train” 账号即可完成)。
* 如果你没有分配到服务器，你可以去 Docker Hub 中 https://hub.docker.com/r/jkjkmxmx/sast2023-linux-git 下载原生的镜像来游玩，该镜像是 x86-64/amd64 架构的，最好保证你的计算机也是此架构 ~~(好像用 arm64 的 QEMU 模拟会出现某些 flag 拿不到的问题)~~。具体流程如下：

```
1 docker pull jkjkmxmx/sast2023-linux-git
2 docker run --privileged -d -p 10000:22 -p 10001:80 -p 10002:3306 -p 10003:10001 -p
  ↪ 10004:10002 -h sast2023 --name sast2023 jkjkmxmx/sast2023-linux-git
3 ssh -p 10000 train@localhost
```

3.6 CTF — Linux 区

该区在 “train” 的家目录 (即 `/home/train`) 下的 “puzzles” 目录中进行。在这个目录中的每一个子目录 (如 “envir”) 就代表一个谜题，**一个谜题恰对应一个 “flag”**。

每个谜题下一定存在一个与目录名同名的可执行文件 (可以有多种形式：二进制文件，Shell 脚本等；以 “envir” 为例，即为 `envir/envir`)，运行该文件并按照它的引导一步步获取 “flag”。

3.7 CTF — Git 区

该区在 “train” 的家目录下的 “git” 目录中进行。本区包含 6 个 flag，简要介绍如下：

3.7.1 Branches

* 看看这个仓库里有哪些分支？

3.7.2 Message

* 提交信息不止只有标题哦，还有正文！

3.7.3 History

* 数据被覆盖了！看看怎么跳回之前的版本？

3.7.4 Reflog

* 该谜题在 “reflog” 分支下进行。
* 观察提交信息：“add correct Taylor formula”，这说明之前有可能加入了错误的公式。
* 但是它不在版本树里，这说明很可能被 `git reset` 过。那该如何恢复这样的 “误操作” 呢？

3.7.5 Werewolf

* 该谜题在 “werewolf” 分支下进行。
* **state 文件下看起来有很多 “flag”，但是这些基本都是假的。**
* “预言家” 告诉你：最终版本的所有 “flag” 中，唯一一个由 “狼人” (werewolf) 提供且没有被后面平民 (villager) 覆盖的 “flag” 才是真正的 “flag”。

3.7.6 Debug

* 该谜题在 “debug” 分支下进行。
* “cat” 对 `debug.cpp` 文件修改了很多次，但这些文件产生运行时错误了！“cat” 想知道，究竟是哪一次提交，使这个程序第一次产生运行时错误呢[12]？
* [该怎么寻找这样的提交呢？二分查找？如何在版本树中进行二分查找？]
* **只有这个提交正文里的 “flag” 才是真正的 “flag” 哦！**

3.8 CTF — Bonus 区

该区会在任何可能的地方进行，它们有可能出现在服务器中，也有可能出现在其他任何地方。作为彩蛋，这里不给出相关的提示，希望大家能开心地寻找。

4 致谢

本次作业参考了 2021 与 2022 年计算机系暑培 Linux 讲义与作业 [1]。本次作业镜像使用计算机系科协提供的 @Zeus 服务器进行分发，感谢它在 2#-308B 日日夜夜勤勤恳恳地工作。本次作业的镜像上传到了 Docker Hub，所编写代码将会在作业结束后全部开源，供后续培训设计时作为参考。

References

[1] c7w. SAST 2022 Summer Training, CTF | A Warmup for Linux. https://github.com/c7w/sast2022-linux-training/blob/master/Linux.pdf, 2022.

[2] Linux manual page. man(1). https://man7.org/linux/man-pages/man1/man.1.html.

[3] SAST skill docers. Git 入门. https://docs.net9.org/basic/git/, 2023.

[4] SAST skill docers. Linux 入门. https://docs.net9.org/basic/linux/, 2023.

[5] 刘轩奇. Linux Basics with Vim, SSH and Shell Language. https://cloud.tsinghua.edu.cn/d/08c22202c3f24cb6abb3/files/?p=/03_Linux_Basics/03_%E8%AF%BE%E4%BB%B6.pptx, 2021.11.

[6] 陈拓. Git 使用教程. https://cloud.tsinghua.edu.cn/d/08c22202c3f24cb6abb3/files/?p=/04_Git/git_instruction.pdf, 2021.11.

[7] 高宇睿. Git 课前准备. https://cloud.tsinghua.edu.cn/d/276d435f204c429c9432/files/?p=/07-14%20Git/07-14-Git-%E8%AF%BE%E5%89%8D%E5%87%86%E5%A4%87.pdf, 2021.7.

[12] 保证存在这样的提交，使得在其之前 `debug.cpp` 正常运行，它及其它之后均产生运行时错误。