w1_ssh

SSH

Secure shell (SSH) is a protocol to allow you to remotely log in to another computer.

`ssh` is the client, which you run on your machine to connect to another machine.

`sshd` is the server, or *daemon* in UNIX-speak. It runs in the background on the machine you want to connect to, and needs to be installed by the system administrator. Note: SSH uses TCP port 22 by default.

Check your client

Type `ssh localhost` and press ENTER. Several different things could happen:

- If it asks for a password, then the ssh client is working, and a ssh server is running on your current machine. The password would be your user account password, but we don't actually want to log in again so cancel with Control+C.
- If it succeeds without a password, then the client is working and a ssh server is running on your machine and either you do not have a password, or you already have a key set up. Type `exit` and press ENTER to get back to your previous shell.
- If it shows "connection refused", then you have the ssh client correctly working but no server running on your own machine. This is not a problem, as we're trying to log in to the lab machines, so we need a client on our machine and a server on the lab machine.
- If it shows an error that ssh is not found, then you don't have (Open)SSH installed which is very unusual except on windows CMD - in which case please switch to using the windows subsystem for linux.

Connect to the lab

`$ ssh [USERNAME@]HOSTNAME`

The bastion host `seis.bris.ac.uk`. This is reachable over SSH from the internet, and is on a university network that lets you connect further to the lab machines. You should not attempt to do any work on seis itself, as most of the software you would like to use (like compilers) is not installed there. However, you do have a home directory on seis for storing things like SSH keys.

The load balancer `rd-mvb-linuxlab.bristol.ac.uk` connects you to a lab machine.

Prompt: `USERNAME@it######:~$`

Type `ssh USERNAME@seis.bris.ac.uk` . Command `uname -a` to print information about the system. Try `whoami` and `uname -a` to check who you are logged in as, and where; also try `hostname` which just prints the machine name.

Jump:

```
ssh -J USERNAME@seis.bris.ac.uk USERNAME@rd-mvb-linuxlab.bristol.ac.uk
```

Setting up ssh keys

The keys that SSH uses implement digital signatures. Each key comes as a pair of files:

A private key (also known as secret key) in a file normally named `id_CIPHER` where CIPHER is the cipher in use. You need to keep this secure and only store it in places that only you have access to.

A public key in a file normally named `id_CIPHER.pub`. You can share this with the world, and you will need to store a copy of it on any machine or with any service that you want to log in to (for the lab, because the lab machines all share a file system, you only need to store it once - but seis has a separate file system so you need a separate copy there).

Generate keys: command `ssh-keygen -t ed25519` .

On your own machine: `ls -l ~/.ssh`

```
-rw-r--r--.  config
-rw-------.  id_ed25519
-rw-r--r--.  id_ed25519.pub
-rw-r--r--.  known_hosts
```

~/.ssh/authorized_keys and ~/.ssh/id_ed25519.pub:

`ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIDxt6V4EEZ+7knCtSjsSYAtomEsH2WStEOQTE2JlwlHL saquantum@localhost.localdomainssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIHQarE6bKBCgqhXXPyGPWRJaf8I8JZNhwHHfmNQQxFQz soyo@DESKTOP-PF2B1AC`

Set up key access on SEIS

First, we need to upload our public key to the `~/.ssh` directory on seis. Even before this, we need to make sure the directory exists though:

- Log in to seis with ssh and your password.
- Try `ls -al ~/.ssh`. If it complains the folder doesn't exist, create it with `mkdir ~/.ssh`.
- Log out of seis again with `exit`.

Run this from your own machine:

```
scp ~/.ssh/id_ed25519.pub "USERNAME@seis.bris.ac.uk:~/.ssh/"
```

This will ask for your password again. Note two things here: first, to set up access on seis, we are uploading the public key - not the private key! - and secondly, that we put double quotes around the destination. This is because the `~` character meaning home directory is handled by our shell, but we don't want our local shell to expand it, instead we want the shell on seis launched by scp to expand it to our home directory on that machine.

Now log in to seis over ssh and type your password one last time. Then run the following:

```
cd .ssh
cat id_ed25519.pub >> authorized_keys
chmod 600 authorized_keys
```

Setting up keys for lab machines

To connect from your machine to seis, you need a private key on your machine and a public key on seis. To connect from seis to a lab machine, it would seem like you need a public key on the lab machine and a private key on seis. You do not want to upload your private key to seis though for security reasons, so instead we are going to use a SSH feature called *agent forwarding* which means that if you SSH into one machine, then when you try and SSH further into another machine SSH will reuse the same key. The way to do this is to use the `-A` command line flag.

To set this up:

- Log in to seis with `ssh USERNAME@seis.bris.ac.uk`. You should not need a password anymore.
- Log in to the lab machines with `ssh rd-mvb-linuxlab.bristol.ac.uk` and enter your password. Check that the `~/.ssh` folder exists and create it if it doesn't, as you did before on seis, then `exit` again to seis.
- Copy your public key file from seis to the lab machines with `scp ~/.ssh/id_ed25519.pub "rd-mvb-linuxlab.bristol.ac.uk:~/.ssh/"`. This will ask for your password again.
- Log in to a lab machine with `ssh rd-mvb-linuxlab.bristol.ac.uk` and enter your password one last time. On the lab machine, install the public key with the following:

```
cd .ssh
cat id_ed25519.pub >> authorized_keys
chmod 600 authorized_keys
```

- Log out of the lab machine and seis again by typing `exit` twice.

The steps above were necessary because your home directory on seis is not the same as on the lab machines.

From now on, from you own machine, you should be able to get directly into a lab machine with the following command, which should not ask for your password at all:

```
ssh -A -J USERNAME@seis.bris.ac.uk USERNAME@rd-mvb-linuxlab.bristol.ac.uk
```

Setting up a configuration file

SSH reads two configuration files: one for all users at `/etc/ssh/ssh_config` (`/etc` is where POSIX programs typically store global settings) and a per-user one at `~/.ssh/config`.

Create `~/.ssh/config` on your own machine:

```
Host seis
  HostName seis.bris.ac.uk
  User USERNAME

Host lab
  HostName rd-mvb-linuxlab.bristol.ac.uk
  ProxyJump seis
  User USERNAME
```

Running vagrant

- Open a terminal in the folder containing the Vagrantfile.
- Run the command `vagrant up`. This starts the virtual machine configured in the current folder, and if it has not been downloaded and provisioned yet (as is the case when you run `up` for the first time) then it does this for you as well.
- When Vagrant tells you the machine is running, run `vagrant ssh` to log in to your virtual machine. If it asks you for a password, use `vagrant`.
- You should now see the virtual machine prompt `vagrant@debian12:~$`. Try the command `ls /` and check that there is a folder called 'shared' in the top-level folder, along with system ones with names like `usr` and `bin`.
  There are two kinds of errors you might get during `vagrant up`:
- If vagrant complains that it can't find a provider, then you have probably not installed virtualbox, or not rebooted since installing it.
- If you get some odd crash or error message about hypervisors: you cannot run vagrant when another program is already using your processor's virtualisation subsystem, and the page gives instructions how to turn off the other one.
  To exit the virtual machine, type `exit` which will get you back to the shell on the host machine. On the host, `vagrant halt` cleanly shuts down the virtual machine.

The file system

Have a look with the command `ls /`:

`/bin` stands for binaries, that is programs that you can run. Have a look with `ls /bin`: there will be a lot of commands in here, including ls itself. Indeed you can find out where a program is with `which`, so `which ls` will show you `/usr/bin/ls` for example.

`/usr` is a historical accident and a bit of a mess. in the earliest days,

- `/bin` was only for binaries needed to start the system - or at least the most important binaries that needed to live on the faster of several disk drives, like your shell.
- `/usr/bin` was where most binaries lived which were available globally, for example across all machines in an organisation.
- `/usr/local/bin` was for binaries installed by a local administrator, for example for a department within an organisation.

In any case, `/usr` and its subfolders are for normally read-only data, such as programs and configuration files but not temporary data or log files. It contains subfolders like `/usr/bin` or `/usr/lib` that duplicate folders in the root directory. Debian's way of cleaning this mess up is to make its `/bin` just a link to `/usr/bin` and putting everything in there, but in some distributions there are real differences between the folders.

If you have colours turned on (which is the default) you will see some files are green, but others are blue - this indicates the file type, green is an executable program, blue is a link to another file. Have a look with `ls -l /bin`: the very first character of each line indicates the file type, the main ones being `-` for normal file, `d` for directory and `l` for a so-called *soft link*. You can see where each link links to at the end of this listing. For example, `slogin` links to `ssh`. Other links point at files stored elsewhere in the filesystem – you'll see a lot of references to `/etc/alternatives/`.

`/etc` stores system-wide configuration files and typically only root (the administrator account) can change things in here. For example, system-wide SSH configuration lives in `/etc/ssh`.

`/lib` contains dynamic libraries - windows calls these `.dll` files, POSIX uses `.so`. For example, `/lib/x86_64-linux-gnu/libc.so.6` is the C library, which allows C programs to use functions like `printf`.

`/home` is the folder containing users' home directories, for example the default user vagrant gets `/home/vagrant`. The exception is root, the administrator account, who gets `/root`.

`/sbin` (system binaries) is another collection of programs, typically ones that only system administrators will use. For example, `fdisk` creates or deletes partitions on a disk and lots of programs with `fs` in their name deal with managing file systems. `/sbin/halt`, run as root (or another user that you have allowed to do this), shuts down the system; there is also `/sbin/reboot`.

`/tmp` is a temporary filesystem that may be stored in RAM instead of on disk (but swapped out if necessary), and that does not have to survive rebooting the machine.

`/var` holds files that vary over time, such as logs or caches.

`/dev`, `/sys` and `/proc` are virtual file systems. One of the UNIX design principles is that almost every interaction with the operating system should look to a program like reading and writing a file, or in short *everything is a file*. For example, `/dev` offers an interface to devices such as hard disks (`/dev/sda` is the first SCSI disk in the system, and `/dev/sda1` the first partition on that), memory (`/dev/mem`), and a number of pseudoterminals or ttys that we will talk about later. `/proc` provides access to running processes; `/sys` provides access to system functions. For example, on some laptop systems, writing to `/sys/class/backlight/acpi_video0/brightness` changes the screen brightness.

The `/shared` folder is not part of the FHS, but is this unit's convention for a shared folder with the host on Vagrant virtual machines. In previous years we called this folder `/vagrant`, but given the default username on the VM is *also* 'vagrant', this led to a lot of confusion, so we changed it.

Package managers

```
sudo apt install PACKAGE
```

- `sudo` (superuser do) allows you to run a command as root, also known as the administrator or superuser. It is good practice to use sudo for system adminstration instead of logging in as root directly, but if you ever really need a root shell then `sudo bash` gets you one - with `#` instead of `$` as prompt to warn you that you are working as root.
- `apt` is the Debian package manager.
- `install PACKAGE` adds a package, which means download and install it and all its dependencies.

You can also find out information about packages with `apt info PACKAGE` .

You can remove them from the system with `sudo apt remove PACKAGE`.

The repositories that you are using are recorded in `/etc/apt/sources.list`, have a look at this file with `cat` to see where they are, then look up the sites in your browser. There are folders for different Debian versions and package index files for different architectures.

Two commands a system adminstrator should run regularly for security reasons:

- `sudo apt update` fetches the new package list from the repository. This way, apt can tell you if any packages have been updated to new versions since you last checked.
- `sudo apt upgrade` upgrades every package that you already have installed to the latest version in your local package list (downloaded when you do an `apt update`).

w2_shell

Bash

Pattern matching

Try the following:

- `./arguments *` in the folder that contains the arguments program.

```
Argument #0: [./arguments]
Argument #1: [arguments]
Argument #2: [arguments.c]
...
Argument #8: [test.c]
```

- Make an empty subfolder with `mkdir empty`, switch to it with `cd empty` and then run `../arguments *`.

```
Argument #0: [../arguments]
Argument #1: [*]
```

- Go back to the folder and find three different ways to get the program to produce the above output.

```
./arguments \*
./arguments "*"
./arguments '*'
```

Files with spaces in their names

Create a file with a space in its name by typing `touch "silly file"`. Start typing `cat sill` and then press TAB to autocomplete:

```
cat silly\ file
```

Use this method to get the arguments program to print the following:

```
./arguments Hello\ world!
Argument #0: [./arguments]
Argument #1: [Hello world!]
```

Shell variables

In the shell, `VARIABLE=VALUE` sets a variable to a value and `$VARIABLE` retrieves its value. For example, to save typing a filename twice:

```
p=arguments
gcc -Wall $p.c -o $p
```

If you want to use a variable inside a word, you can use curly braces: `${a}b` means the value of the variable `a` followed by the letter b, whereas `$ab` would mean the value of the variable `ab`.

It is good practice to double-quote variables used like this, because if you tried for example to compile a program called `silly name.c` with a space in its name, then

```
program="silly name"
gcc -Wall $program.c -o $program
```

would expand to

```
gcc -Wall silly name.c -o silly name
gcc: error: silly: No such file or directory
gcc: error: name.c: No such file or directory
gcc: error: name: No such file or directory
gcc: fatal error: no input files
compilation terminated.
```

Correct would be:

```
program="silly name"
gcc -Wall "$program.c" -o "$program"
```

which expands to

```
gcc -Wall "silly name.c" -o "silly name"
```

Note that this does not work as expected either:

```
file=arguments gcc -Wall "$file.c" -o "$file"
```

The problem here is that the shell first reads the line and substitutes in the value of `$file` (unset variables expand to the empty string by default) before starting to execute the command, so you are reading the variable's value before writing it. Leaving off the quotes doesn't help: you need to set the variable on a separate line.

Create a user and a group

Create a new user with `sudo adduser NAME` . Check the user and group files with `tail /etc/passwd` and `tail /etc/group` to check that the new user has been created.

Time to change user: `su brian` and enter the password. Notice that the prompt has changed to `brian@debian12:/home/vagrant$`. So the user has changed, and because `/home/vagrant` is no longer the current user's home directory, it gets written out in full.

Next, create a user `nigel` (or some other name) add both your two new users, but not `vagrant`, to the group `users` (which already exists) using the command `sudo usermod -aG GROUPNAME USERNAME`.

Remove user from a group:

```
sudo gpasswd -d vagrant users
```

Explore file permissions

As user `brian` set up your home directory so that

- You can do everything (rwx).
- Members of the `users` group can list files and change to your home directory, but not add/remove files. You will need to change the group of your home directory to `users` for this, using the command `chgrp -R GROUPNAME DIRECTORY`.
- Everyone else cannot do anything with your home directory.

  ```
  chmod 750 ~
  ```

Create a file in your home directory, check, by using `su USERNAME` to log in as the different users, that:

- `nigel` can view Brian's home directory but not create files there;
- `nigel` can view but not edit Brian's readme file;
- `vagrant` cannot list files in or enter Brian's home directory at all. What happens when you try?

Also as `brian`, make a `private` subdirectory in your home folder that no-one but you can access. Create a file `secret.txt` in there with `nano private/secret.txt` as user `brian` from Brian's home directory. Check as Nigel that you can see the folder itself, but not cd into it nor list the file.

Using `ls -l` as Brian in both `~` and `~/private`, compare the entries for the files `~/readme.txt`, `~/private/secret.txt` and the folder `~/private`. Why do the groups of the two files differ? The rule is that you need permissions on the whole path from `/` to a file to be able to access it.

Setuid

As Brian, create a file `message-brian.c` in your home directory. Compile it and run `chmod u+s message-brian` and check the file again: you should now see `-rwsr-xr-x` for the file permissions. The `s` is the setuid bit. **Setuid (Set User ID)** is a special permission bit in Unix/Linux file systems that allows a file to be executed with the **privileges of its owner**, rather than the user who runs it. This is particularly useful for programs that need elevated permissions to perform specific tasks.

As Nigel go into Brian's home directory and run `./message-brian "Hi from Nigel!"`. Now run `ls -l` and notice that a `messages.txt` has appeared with owner and group `brian`. Although Nigel cannot create and edit files in Brian's home directory himself (he can't edit `messages.txt` for example, although he can read it), the program `message-brian` ran as Brian, which let it create the file.

```
brian@debian12:~$ gcc -Wall message.c -o message
brian@debian12:~$ ls -l
total 20
-rwxr-xr-x 1 brian brian 16176 Nov 30 18:26 message
-rw-r--r-- 1 brian brian   542 Nov 30 18:26 message.c
brian@debian12:~$ chmod u+s message
brian@debian12:~$ ls -l
total 20
-rwsr-xr-x 1 brian brian 16176 Nov 30 18:26 message
-rw-r--r-- 1 brian brian   542 Nov 30 18:26 message.c
brian@debian12:~$ su nigel
Password:
nigel@debian12:/home/brian$ ./message "Hi from n"
nigel@debian12:/home/brian$ ls -l
total 24
-rwsr-xr-x 1 brian brian 16176 Nov 30 18:26 message
-rw-r--r-- 1 brian brian   542 Nov 30 18:26 message.c
-rw-r--r-- 1 brian nigel    10 Nov 30 18:27 messages.txt
nigel@debian12:/home/brian$ cat messages.txt
Hi from n
```

Finding all files with the setuid bit set on a system:

```
sudo find / -perm /4000
```

You might get some errors relating to `/proc` files, which you can ignore: these are subprocesses that find uses to look at individual files.

Apart from `message-brian`, you'll find a few files by default: `sudo`, `mount`, `umount` and `su`. The first one you already know; look up what the next two do and think about why they are setuid. Specifically, what kinds of (un)mounting are non-root users allowed to do according to the manual pages?

Look up the `passwd` program in the manual pages. Why might that program need to be setuid?

```
The passwd command needs to have the setuid permission because it modifies sensitive system files, su
ch as /etc/passwd and /etc/shadow, which are typically only writable by the root user.

By setting the setuid bit, the passwd program runs with the privileges of its owner (usually root), r
egardless of which user executes it. This allows regular users to change their passwords securely wit
hout giving them unnecessary root access to the system.
```

Sudo

Switch back to `vagrant` and run the command `sudo cat /etc/sudoers`. Everything is commented out except `root ALL=(ALL) ALL` and the last line `#includedir /etc/sudoers.d` (this is not a comment!) which contains a single file `vagrant` with the line `vagrant ALL=(ALL) NOPASSWD: ALL` which is why vagrant can use sudo in the first place.

However, note the commented lines such as

```
# %wheel ALL=(ALL) NOPASSWD: ALL
# %sudo ALL=(ALL) ALL
```

If uncommented, the first one would let everyone in group wheel run commands using sudo (this is the default on some other Linux distributions), whereas the second one would allow everyone in the group `sudo` to do this, but would prompt for their own password beforehand.

Open a root shell with `sudo su` as user `vagrant`; this is so we don't get locked out if we break sudo. Edit the sudoers file with `visudo` as root, and add the following line:

```
%users ALL=(ALL) /sbin/reboot
```

You can now switch back to `brian` and do `sudo reboot`.

Shell Scripting

For portable POSIX shellscripts `#! /bin/sh/`
For less portable BASH scripts

`#! /usr/bin/env bash`

```
chmod +x script.sh
./script.sh
```

Programs return a 1 byte exit value. This gets stored into the variable ${?} after every command runs.

```
test $? -eq 0 && echo "Command succeeded\n"
[ $? -eq 0 ] && echo "Command succeeded\n"
```

variables:

```
p="Hello World"
echo "${p}"
export p
unset p
echo "${p}"
set -o nounset
echo "${p:? unset variable}"
```

`${0}` Name of the script

`${1}, ${2}, ${3}…` Arguments passed to your script

`${#}` The number of arguments passed to your script

`${@}` and `${*}` All the arguments

`$(COMMAND)` use to capture the output of command

`$((ARITHMETIC))` use to perform arithmetic operations

`if` statement

```
if CONDITION1; then

elif CONDITION2; then

else

fi
```

- File Checks:
  - `[ -f FILE ]`: True if FILE exists and is a regular file.
  - `[ -d DIR ]`: True if DIR exists and is a directory.
  - `[ -e FILE ]`: True if FILE exists (any type of file).
  - `[ -x FILE ]`: True if FILE exists and is executable.
- String Checks:
  - `[ "$STR" = "value" ]`: True if `STR` is equal to `"value"`.
  - `[ -z "$STR" ]`: True if `STR` is empty.
  - `[ -n "$STR" ]`: True if `STR` is not empty.
- Numeric Comparisons:
  - `[ "$A" -eq "$B" ]`: True if `A` is equal to `B`.
  - `[ "$A" -ne "$B" ]`: True if `A` is not equal to `B`.
  - `[ "$A" -gt "$B" ]`: True if `A` is greater than `B`.
  - `[ "$A" -lt "$B" ]`: True if `A` is less than `B`.
  - `[ "$A" -ge "$B" ]`: True if `A` is greater than or equal to `B`.
  - `[ "$A" -le "$B" ]`: True if `A` is less than or equal to `B`.

`for` statement

```
for VARIABLE in LIST; do

done
```

```
for n in 1 2 3 4 5; do
echo -n "${n} "
done

for n in $(seq 5); do
echo -n "${n} "
done

IFS=','
for n in $(seq -s, 5); do
echo -n "${n} "
done
```

`case` statement

```
case VARIABLE in
    PATTERN1)
        ;;
    PATTERN2)
        ;;
    *)
        ;;
esac
```

`basename` and `dirname`

```
basename [path] [suffix]
```

`path`: The full file path.

`suffix` (optional): A string to remove from the end of the filename (e.g., a file extension).

```
dirname [path]
```

Compile helper exercise

Building on what you saw in the videos and the previous exercises, write a shell script in a file called `b` (for build) that does the following:

- Your script should run under any Bourne-compatible shell (e.g. not just `bash`), and it should be written so that you can call it with `./b` .
- `./b compile NAME` should compile the file of the given name, so for example `./b compile hello` should run `gcc -Wall -std=c11 -g hello.c -o hello`.
- However, your script should accept both `./b compile hello` and `./b compile hello.c` as input, and do the same thing in both cases, namely compile `hello.c`. The output file for gcc in both cases should be called just `hello`.
- If the source file you provided as an argument does not exist (adding `.c` if necessary) then the script should print an error message and return a nonzero exit status - *not* invoke the C compiler.
- `./b run NAME` should run the program, assuming it exists in the current folder, so both `./b run hello` and `./b run hello.c` should run `./hello`. If it does not exist, again print an error message and exit with a nonzero status, don't try and run the program.
- `./b build NAME` should first compile the C source file, and then if the compile was successful it should run the program. If the compile failed, it should not try and run the program.
- If you call `./b` without any parameters, or `./b COMMAND` with a command other than compile or run or build, it should print some information on how to use it. If you call `./b compile` or another command with no filename at all, then the script should print an error message and exit with a nonzero exit status.

  ```
  #! /usr/bin/env bash

  if [ ${#} -eq 2 ];then
  ```

```
    NAME=$(basename ${2} .c)
    if [ ! -f "${NAME}.c" ];then
        echo "non-existing file"
        exit 1
    fi

    if [ "${1}" = "compile" ];then
        gcc -Wall -std=c99 "${NAME}.c" -o "${NAME}"
        exit 0
    fi

    if [ "${1}" = "run" ];then
        ./"${NAME}"
        exit 0
    fi

    if [ "${1}" = "build" ];then
        gcc -Wall -std=c99 "${NAME}.c" -o "${NAME}"
        ./"${NAME}"
        exit 0
    fi
fi

echo "run the bash with \"./b build FILENAME.\" "
```

Strict Mode

Use the following line near the top of your shell scripts: `set -euo pipefail`

- `set -e` makes the whole script exit if any command fails.
- `set -u` means referencing an undefined variable is an error.
- `set -o pipefail` changes how pipes work: normally, the return value of a pipe is that of the *last* command in the pipe. With the `pipefail` option, if any command in the pipeline fails (non-zero return) then the pipeline returns that command's exit code.

Pipes

- `cat [FILENAME [FILENAME...]]` writes the contents of one or more files to standard output. This is a good way of starting a pipe. If you leave off all the filenames, cat just reads its standard input and writes it to standard output.
- `head [-n N]` reads its standard input and writes only the first N lines (default is 10 if you leave the option off) to standard output. You can also put a minus before the argument e.g. `head -n -2` to *skip the last 2 lines* and write all the rest.
- `tail [-n N]` is like head except that it writes the last N lines (with a minus, it skips the first N ones).
- `sort` reads all its standard input into a memory buffer, then sorts the lines and writes them all to standard output. `r` reverse order.
- `uniq` reads standard input and writes to standard output, but skips repeated lines that immediately follow each other, for example if there are three lines A, A, B then it would only write A, B but if it gets A, B, A it would write all three. A common way to remove duplicate lines is `... | sort | uniq | ...`.
- `wc [-l]` stands for word count, but with `l` it counts lines instead. `w` count words.
- `command | tee [-a] [file...]` . copy the output of command to file as well as standard output. `a` append instead of overwriting.

  `0` standard input

  `1` standard output

  `2` standard error `out 2> file`

  `|` standard output → standard input

  `>` standard output → file (overwrite)

  `>>` standard output append to file

  `<` file → standard input

  `<<<` string as temporary file to standard input

  `2>&1` merge standard error into standard output `ls nonexistingfile > output 2>&1`

If PROGRAM needs a file to read from but pipe in (subshell):

```
PROGRAM <(SOMETHING)
cat <(echo "Hi")
```

subshell to argument:

```
COMMAND $(SOMETHING)
echo $(ls -l)
```

Word list exercises

dictionary file `/usr/share/dict/words`

- The number of words in the words file - there is one word per line.

  ```
  cat /usr/share/dict/words | wc -l
  ```

- The 6171st word in the file.

  ```
  head -n 6171 /usr/share/dict/words | tail -n 1
  ```

- The first five words that are among the last 100 words on the list.

  ```
  tail -n 100 /usr/share/dict/words | head -n 5
  ```

- The last ten words in the file, sorted in reverse order.

  ```
  tail -n 10 /usr/share/dict/words | sort -r
  ```

Redirection

Find a directory on your VM that contains some files, and is writeable by your current user.

- `ls -l` prints a detailed listing of the directory to standard output. Redirect this listing to instead store it in a file called `tmp`.

  ```
  ls -l > tmp
  ```

- If you inspect the file content (e.g., by `cat tmp`) and compare it to what you see when re-typing `ls -l` now, you should be able to find a difference. The `diff` utility takes in (at least) two filenames and produces output that compares the two files.

  ```
  diff tmp <(ls -l)
  ```

- Redirect it to a file `difflog`. By creating this file, we're changing what `ls` will report about this directory. Let's keep track of that change by running the same `diff` command again, but this time *appending* the new output to `difflog`.

  ```
  diff temp <(ls -l) >> difflog
  ```

w3_regex

Regular Expression

grep (Global Regular Expression Print)

dictionary file `/usr/share/dict/words`

```
grep [-option] REGEX
```

options:

`-i` case-insensitive

`-v` invert match, that do not match the expression

`-E` extended regex

`-r` recursively search through directories

`-c` count lines

`-w` match whole word

`-o` only display the matched text and output each match in separate lines

REGEX:

`^` begin

`$` end

`[^ ]` exclude

`.` any character

`?` 0 or 1

`*` 0 or more

`+` 1 or more

`{n}` exactly n

`{m,n}` m or n

exercise

All words containing the letter capital Q.

```
    grep Q
```

All words starting with the letter R, in either upper or lower-case.

```
    grep -i R
```

All words ending in j.

```
    grep j$
```

The number of words containing the letter Q, ignoring case.

```
    grep -ic Q
    grep -i Q | wc -l
```

The first five words containing the letter sequence ‘cl’.

```
    grep cl | head -n 5
```

All words containing the sequence “kp”, but not “ckp”.

```
    grep [^c]kp
```

The last 15 words of exactly two letters.

```
grep ^..$ | tail -n 15
```

All three-letter words with no vowels (aeiou).

```
grep -iE ^[^aeiou]{3}$
```

All words of exactly 7 letters, where the third one is an e and the word ends “-ded”.

```
grep ^..e.ded$
```

Find all words that start with a P (whether capitalised or not), and contain at least four instances of the letter a.

```
grep ^[pP].*a.*a.*a.*a.*$
```

Contrive a file such that `grep` returns multiple lines but `grep -w` returns only one line.

```
echo -e "cat\ncatalog\ncater\nscatter" | grep -w 'cat'
```

find a situation where `grep -o PATTERNFILE | wc -l` and `grep -c PATTERNFILE` produce different results

```
echo -e "one cat two cat\none cat" | grep -c cat -> 2
echo -e "one cat two cat\none cat" | grep -o cat | wc -l -> 3
```

match both ‘encyclopaedia’ and ‘encyclopedia’ but nothing else.

```
grep -wE 'encyclopa?edia'
```

match UK postcodes.

```
grep -E [A-Z]{2}[0-9][\ ]?[0-9][A-Z]{2}
grep -E '[A-Z]{2}[0-9] ?[0-9][A-Z]{2}'
```

find an example that would match the following but fail to match the above.

```
^(
    ([A-Z]{1,2}[0-9][A-Z0-9]?|ASCN|STHL|TDCU|BBND|[BFS]IQQ|PCRN|TKCA) ?[0-9][A-Z]{2}
    |BFPO ?[0-9]{1,4}
    |(KY[0-9]|MSR|VG|AI)[ -]?[0-9]{4}
    |[A-Z]{2} ?[0-9]{2}
    |GE ?CX
    |GIR ?0A{2}
    |SAN ?TA1
)$
```

sed (Stream Editor)

```
sed [options] 's/SOURCE/DEST/'
```

options:

`-e` multiple commands in one line

`-E` extended regex

`'s/SOURCE/DEST/'`

`\( \)` create group in SOURCE to be referred in DEST

exercise

find all words ending in ‘ay’ and change ‘day’ into ‘week’.

```
grep ay$ | sed s/day/week/
```

In the same selection as above, replace all words that begin with ‘s’ with the word ‘sway’.

```
grep ay$ | sed s/^s/sway/
```

duplicate the match after a space, for any line containing ‘day’, so “saturday” becomes “saturday day”.

```
grep ay$ | sed 's/day/& &/'
```

any line ending in ‘day’ becomes a string “Xday or Xweek”, where X is the other part of the word.

```
grep ay$ | sed 's/\(.*\)day$/\1day or \1week/'
```

any word ending in either ‘way’ or ‘day’ to be flipped around and parenthesised, so ‘someday’ becomes ‘day (some)’ and ‘speedway’ becomes ‘way (speed)’.

```
grep ay$ | sed 's/\(.*\)\([dw]ay\)/\2 (\1)/'
```

difference between applying `s/a/e/` and `s/a/e/g`:

without g command only replace the first occurrence. `echo "banana" | sed 's/a/e/'` → “benana”.

with g command replace all occurrences. `echo "banana" | sed 's/a/e/g'` → “benene”.

w4_git

Git

Configuring your identity

```
git config --global user.name "YOURNAME"
git config --global user.email "YOUREMAIL"
```

A sample project and repository

```
$ mkdir gitproject
$ cd gitproject
$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /home/kj24716/gitproject/.git/
$ git log
fatal: your current branch 'master' does not have any commits yet
$ echo -e "#include<stdio.h>\nint main(){\nprintf(\"Hi\");\nreturn 0;\n}\n" > test.c
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        test.c

nothing added to commit but untracked files present (use "git add" to track)
$ git add test.c
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   test.c

$ git commit -m "first file"
[master (root-commit) 4e87884] first file
 1 file changed, 6 insertions(+)
 create mode 100644 test.c
$ git status
On branch master
nothing to commit, working tree clean
$ git log
commit 4e878843247de9b4ced79406b1ea97c5f19b7564 (HEAD -> master)
Author: saquantum <kj24716@bris.ac.uk>
Date:   Sun Dec 1 02:02:33 2024 +0000

    first file
```

Ignoring files

- Create a file `.gitignore` and add the single line `program` to it.
- Do another `git status` and notice that while the program is now ignored, the ignore file is marked as new. This file does belong in the repository, so add it and commit it.

```
$ gcc test.c -o test
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        test

nothing added to commit but untracked files present (use "git add" to track)
$ echo "test" > .gitignore
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore

nothing added to commit but untracked files present (use "git add" to track)
$ git add .
$ git commit -m "ignore file"
[master bc20bbf] ignore file
 1 file changed, 1 insertion(+)
 create mode 100644 .gitignore
$ git log
commit bc20bbf7ddd24faf04aeb44e34b22bdd2f9af259 (HEAD -> master)
Author: saquantum <kj24716@bris.ac.uk>
Date:   Sun Dec 1 02:07:02 2024 +0000

    ignore file

commit 4e878843247de9b4ced79406b1ea97c5f19b7564
Author: saquantum <kj24716@bris.ac.uk>
Date:   Sun Dec 1 02:02:33 2024 +0000

    first file
```

Commit and checkout

Change *Hi* to *Hello* in the program, rebuild and run the program.

```
$ cat test.c | sed 's/Hi/Hello/' > test.c
$ git add .
$ git commit -m "Hi -> Hello"
[master 5e67efd] Hi -> Hello
 1 file changed, 1 insertion(+), 1 deletion(-)
$ git log
commit 5e67efd414a9dbf1c80fa09b6f339cd520768b9e (HEAD -> master)
Author: saquantum <kj24716@bris.ac.uk>
Date:   Sun Dec 1 02:16:36 2024 +0000

    Hi -> Hello

commit bc20bbf7ddd24faf04aeb44e34b22bdd2f9af259
Author: saquantum <kj24716@bris.ac.uk>
Date:   Sun Dec 1 02:07:02 2024 +0000

    ignore file
```

Sometimes you want to go back and look at another commit, or undo a commit that broke something—this is when you want a checkout.

- Note the first 6 or so characters of the commit hash of the commit where you added the ignore file, but before changing *Hi* to *Hello*. You need at least 6 characters, but only as many so that it's not ambiguous to git which commit you mean.
- Run `git checkout HASH` where HASH is the 6 or however many you need characters of the commit in question. Git will print a warning about the HEAD pointer.
- Check the source file, and notice that it is now back on *Hi*.
- Use `git checkout main` to return to the latest version of your files, and git will set up the HEAD pointer again ready to accept new commits.

```
$ git checkout bc20bb
Note: switching to 'bc20bb'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at bc20bbf ignore file
$ git checkout master
Previous HEAD position was bc20bbf ignore file
Switched to branch 'master'
```

HEAD is a pointer to the last commit you checked out: Either the last commit or the place you explicitly checked out with the git checkout command. Work you haven't committed can't be pointed to.

If you actually want to undo a commit, then you have two options:

- `git revert HASH` adds a new commit that returns the files to the state they were before the commit with the given hash. This is safe to use during team development, as it's just adding a new commit. If you have commits A, B and do `git revert B` then you get a new commit C so anyone else using the repository sees a sequence of commits A, B, C; but the state of the files in C is the same as in A.

```
$ git revert 5e67ef
[master 828eb92] Revert "Hi -> Hello"
 1 file changed, 1 insertion(+), 1 deletion(-)
$ git log
commit 828eb929d0450811c00d1f9233bc7880acc77b86 (HEAD -> master)
Author: saquantum <kj24716@bris.ac.uk>
Date:   Sun Dec 1 02:23:19 2024 +0000

    Revert "Hi -> Hello"

    This reverts commit 5e67efd414a9dbf1c80fa09b6f339cd520768b9e.

commit 5e67efd414a9dbf1c80fa09b6f339cd520768b9e
Author: saquantum <kj24716@bris.ac.uk>
Date:   Sun Dec 1 02:16:36 2024 +0000

    Hi -> Hello

$ cat test.c
#include<stdio.h>
int main(){
printf("Hi");
```

```
return 0;
}
```

- `git reset HASH` undoes commits by moving the HEAD pointer back to the commit with the given hash, but leaves the working copy alone (you can use the `-hard` option to change the files as well). This will break things if you have shared your newer commits with other developers, but it's safe to use to undo changes that you haven't pushed yet (we'll learn about this next time). The effect is as if the commits which you've reset had never happened.

```
$ git reset 5e67ef
Unstaged changes after reset:
M       test.c
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   test.c

no changes added to commit (use "git add" and/or "git commit -a")
$ git add .
$ git commit -m "reset to 5e67ef"
[master 30ba9c5] reset to 5e67ef
 1 file changed, 1 insertion(+), 1 deletion(-)
$ git log
commit 30ba9c552322b0ce9e93d60a72a8dcf04d3bfc97 (HEAD -> master)
Author: saquantum <kj24716@bris.ac.uk>
Date:   Sun Dec 1 02:26:11 2024 +0000

    reset to 5e67ef

commit 5e67efd414a9dbf1c80fa09b6f339cd520768b9e
Author: saquantum <kj24716@bris.ac.uk>
Date:   Sun Dec 1 02:16:36 2024 +0000

    Hi -> Hello

commit bc20bbf7ddd24faf04aeb44e34b22bdd2f9af259
Author: saquantum <kj24716@bris.ac.uk>
Date:   Sun Dec 1 02:07:02 2024 +0000

    ignore file
```

Git forges

Click the SSH tab and copy the URL there—it should be something like `git@github.com:USERNAME/REPONAME.git`.

```
$ ssh -T git@github.com
Hi saquantum! You've successfully authenticated, but GitHub does not provide shell access.
$ git clone git@github.com:saquantum/testgit.git
```

Go to that folder, and try `git remote show origin`.

```
$ git remote
origin
$ git remote -v
origin  git@github.com:saquantum/testgit.git (fetch)
origin  git@github.com:saquantum/testgit.git (push)
$ git remote show origin
* remote origin
  Fetch URL: git@github.com:saquantum/testgit.git
  Push  URL: git@github.com:saquantum/testgit.git
  HEAD branch: main
  Remote branches:
```

```
    develop   tracked
    main      tracked
    yzhbranch tracked
    zrbranch  tracked
  Local branch configured for 'git pull':
    main merges with remote main
  Local ref configured for 'git push':
    main pushes to main (up to date)
```

w5_git2

Working with others in Git

`git pull` = `git fetch` + `git merge`

Practice the push workflow

no valid ssh key:

```
$ git clone git@github.com:saquantum/testgit.git
Cloning into 'testgit'...
The authenticity of host 'github.com (20.26.156.215)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3wvvV6TuJJhbpZisF/zLDA0zPMSvHdkr4UvCOqU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
git@github.com: Permission denied (publickey).
fatal: Could not read from remote repository.

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

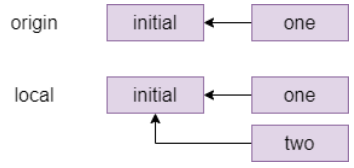
```
$ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Unpacking objects: 100% (3/3), 925 bytes | 132.00 KiB/s, done.
From github.com:Cailian-Liu/GitStudy
   ce4b714..49ecf0e  main       -> origin/main

$ git status
On branch main
Your branch is behind 'origin/main' by 1 commit, and can be fast-forwarded.
  (use "git pull" to update your local branch)

nothing to commit, working tree clean

$ git pull
Updating ce4b714..49ecf0e
Fast-forward
 GitStudy1.txt | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)

$ git add .
$ git commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 GitStudy2.txt

$ git fetch
From github.com:Cailian-Liu/GitStudy
   3b7071f..48675d0  main       -> origin/main

$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

$ git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
```

```
Delta compression using up to 16 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 279 bytes | 279.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:Cailian-Liu/GitStudy.git
   48675d0..b58135f  main -> main
```

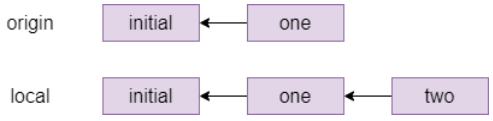
Resolve a fake conflict - rebase

One member adds or changes one file, then commits this change and pushes it by running the whole push workflow. At the same time, the second member adds or changes a different file, then commits this change. The second member starts the push workflow with `git fetch`, then `git status`. Notice you have `diverged`. (If you were to try to `git push`, with or without fetching this would produce an error.)

The commit graph of member two looks something like this:



One way to resolve this conflict is a *rebase*, which is pretending that member two had actually fetched the `one` commit before starting their own work. The command for this which member two types is `git rebase origin/main` which means *pretend that everything in origin/main happened before I started my local changes* and gives the following graph:



If member two does a `git status` after the rebase, they will see `ahead of origin/main by 1 commit` and they can now `git push` to send their local changes to the remote repository.

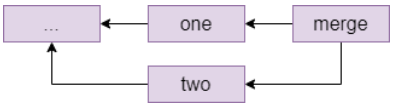
```
$ git fetch
From github.com:Cailian-Liu/GitStudy
   3b7071f..48675d0  main       -> origin/main

$ git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" if you want to integrate the remote branch with yours)

nothing to commit, working tree clean

$ git rebase origin/main
Successfully rebased and updated refs/heads/main.

$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

$ git log
commit b58135facfbe440c84ac6f360f22af4bdbf2b049 (HEAD -> main)
Author: Cailian Liu <liucl2017@foxmail.com>
Date:   Sun Dec 1 11:23:34 2024 -0500

    GitStudy2

commit 48675d01e7ae946a602700bee199e7b7423456b1 (origin/main, origin/HEAD)
```

```
Author: soyo <KatieIove@outlook.com>
Date:   Sun Dec 1 16:22:36 2024 +0000

    abc

$ git push
To github.com:Cailian-Liu/GitStudy.git
   48675d0..b58135f  main -> main
```

Fake conflicts - merge

Like before, team member one edits one file, commits it and does the whole push workflow. The second team member at the same time, without another fetch, edits a different file and commits. The second team member starts the push workflow: fetch, status - notice you've `diverged`.

The second team member types `git pull`. Since this is a fake conflict (different files), this gets you into your editor, and you can see that on the first line is a suggested commit message starting with `Merge branch main`, which it is conventional to accept without changes - exit your editor again. Git replies `Merge made by the recursive strategy.`.




```
$ git fetch
From github.com:Cailian-Liu/GitStudy
   b58135f..7c9d977  main       -> origin/main

$ git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" if you want to integrate the remote branch with yours)

nothing to commit, working tree clean

$ git pull
Merge made by the 'ort' strategy.
 GitStudy1.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

$ git push
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:Cailian-Liu/GitStudy.git
   7c9d977..29de273  main -> main
```

Resolving a real conflict

Team member one creates a file called `README.md` or edits it if it already exists, and adds a line like `Created by NAME` with their own name. Then they commit this change and run the push workflow: `git fetch`, `git status`, check they're `ahead`, `git push` to the remote. Team member two, without fetching the latest commit, creates the same `README.md` file and adds a line `Created by NAME2` and commits this to their local repository. Team member two starts the push workflow: `git fetch`, `git status` and notice that you have `diverged` again.

Run `git pull` as member two. You should see the following message:

```
CONFLICT (add/add): Merge conflict in README.md
Auto-merging README.md
Automatic merge failed; fix conflicts and then commit the result.
```

Open the file and notice that git has annotated it:

```
<<<<<<< HEAD
Created by NAME2.
=======
```

```
Created by NAME1.
>>>>>>> b955a75c7ca584ccf0c0bddccbcde46f445a7b30
```

The lines between `<<<<<<< HEAD` and `=======` are the local changes (team member two) and the ones from `======` to `>>>>>> ...` are the ones in the commit fetched from the remote, for which the commit id is shown. Member two now has to resolve the conflict by editing the file to produce the version they would like to commit. For example, you could remove all the offending lines and replace them with `Created by NAME1 and NAME2.`

Member two can now do the following:

- `git add README.md` (or whatever other files were affected).
- `git commit`. You could give a message directly, but a commit without a `m` drops you into your editor and you'll see that git is suggesting `Merge branch main ...` as a default message here. It is conventional to leave this message as it is, just exit your editor without any changes.
- Run another push workflow: `git fetch`, `git status` and notice you are now `ahead by 2 commits`: the first one was the work you did, the second is the merge commit. You're ahead, so finish the workflow with `git push`.

```
$ git fetch
From github.com:Cailian-Liu/GitStudy
   29de273..86fca56  main       -> origin/main

$ git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" if you want to integrate the remote branch with yours)

nothing to commit, working tree clean

$ git pull
Auto-merging GitStudy2.txt
CONFLICT (content): Merge conflict in GitStudy2.txt
Automatic merge failed; fix conflicts and then commit the result.

$ git fetch

$ git status
On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" if you want to integrate the remote branch with yours)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
        both modified:   GitStudy2.txt

no changes added to commit (use "git add" and/or "git commit -a")

$ git log
commit 9ab7e924dd68008e9e41d70cd14218578d9e9376 (HEAD -> main)
Author: Cailian Liu <liucl2017@foxmail.com>
Date:   Sun Dec 1 11:38:13 2024 -0500

    change by lcl

commit 29de2735995e4a798d1f015d25c3fd605761baa2
Merge: 5b876b4 7c9d977
Author: Cailian Liu <liucl2017@foxmail.com>
Date:   Sun Dec 1 11:33:59 2024 -0500

    Merge branch 'main' of github.com:Cailian-Liu/GitStudy
```

```
commit 5b876b4fb5ad833d715e7a14a4e6d1be3eced6ee
Author: Cailian Liu <liucl2017@foxmail.com>
Date:   Sun Dec 1 11:32:49 2024 -0500

    third change

$ git add .

$ git commit
hint: Waiting for your editor to close the file... unix2dos: converting file C:/Users/lcl/GitStudy/.g
it/COMMIT_EDITMSG to DOS format...
dos2unix: converting file C:/Users/lcl/GitStudy/.git/COMMIT_EDITMSG to Unix format...
[main ac64caa] Merge branch 'main' of github.com:Cailian-Liu/GitStudy

$ git push
To github.com:Cailian-Liu/GitStudy.git
   86fca56..ac64caa  main -> main
```

Working with others

Set-up

One member makes a Git repository on one of the online providers, adds the other team members and shares the cloning URL. Everyone clones the repository.

The repository must have at least one commit for the following to work. This condition is satisfied if you chose your provider's option to create a readme file; if not then make a file now, commit it and push.

The develop branch

By default, your repository has one branch named `main`. But you don't want to do your work on this branch directly. Instead, one team member creates a `develop` branch with the command

```
git checkout -b develop
```

The team member who created the develop branch should now make a commit on it.

This branch currently exists only in their local repository, and if they try and push they would get a warning about this. What they need to do is

```
git push --set-upstream origin develop
```

This adds an "upstream" entry on the local develop branch to say that it is linked to the copy of your repository called `origin`, which is the default name for the one you cloned the repository from.

You can check this with `git remote show origin`, which should display among other things:

```
Remote branches:
  develop tracked
  main  tracked
Local branches configured for 'git pull':
  develop merges with remote develop
  main  merges with remote main
Local refs configured for 'git push':
  develop pushes to develop (up to date)
  main  pushes to main  (up to date)
```

Everyone else can now `git pull` and see the branch with `git branch -a`, the `-a` (all) option means include branches that only exist on the remote. They can switch to the develop branch with `git checkout develop`, which should show:

```
Branch 'develop' set up to track remote branch 'develop' from 'origin'.
Switched to a new branch 'develop'
```

Feature branches

Since everyone is working on a different branch, you will never get a conflict this way. Anyone who is a project member can visit the github page can see all the feature branches there, but a normal `git branch` will not show other people's branches that you've never checked out yourself. Instead, you want to do `git branch -a` again that will show you all the branches, with names like `remotes/origin/NAME` for branches that so far only exist on the origin repository. You can check these out like any other branch to see their contents in your working copy.

```
$ git checkout -b lclbranch
Switched to a new branch 'lclbranch'

$ git add .
$ git commit
$ git fetch
From github.com:Cailian-Liu/GitStudy
 * [new branch]      yzhbranch  -> origin/yzhbranch

$ git status
On branch lclbranch
nothing to commit, working tree clean

$ git push
fatal: The current branch lclbranch has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin lclbranch

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

$ git push --set-upstream origin lclbranch
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'lclbranch' on GitHub by visiting:
remote:      https://github.com/Cailian-Liu/GitStudy/pull/new/lclbranch
remote:
To github.com:Cailian-Liu/GitStudy.git
 * [new branch]      lclbranch -> lclbranch
branch 'lclbranch' set up to track 'origin/lclbranch'.

$ git remote show origin
* remote origin
  Fetch URL: git@github.com:Cailian-Liu/GitStudy.git
  Push  URL: git@github.com:Cailian-Liu/GitStudy.git
  HEAD branch: main
  Remote branches:
    lclbranch tracked
    main      tracked
    yzhbranch tracked
  Local branches configured for 'git pull':
    lclbranch merges with remote lclbranch
    main      merges with remote main
  Local refs configured for 'git push':
    lclbranch pushes to lclbranch (up to date)
    main      pushes to main      (up to date)

$ git branch -a
* main
  remotes/origin/HEAD -> origin/main
  remotes/origin/lclbranch
  remotes/origin/main
  remotes/origin/yzhbranch
```

Merging

When a feature is done, you want to merge it into develop. Everyone should try this, the procedure for this is

1. Commit all your changes and push.
2. Fetch the latest changes from origin (a simple `git fetch` does this).
3. `git checkout develop`, which switches you to the develop branch (the changes for your latest feature will disappear in the working copy, but they're still in the repository). You always merge into the currently active branch, so you need to be on `develop` to merge into it.
4. `git status` to see if someone else has updated develop since you started your feature. If so, then `git pull` (you will be *behind* rather than *diverged* because you have not changed develop yourself yet).
5. `git merge NAME` with the name of your feature branch.
6. Resolve conflicts, if necessary (see below).
7. `git push` to share your new feature with the rest of the team.

```
$ git checkout yzhbranch
branch 'yzhbranch' set up to track 'origin/yzhbranch'.
Switched to a new branch 'yzhbranch'

$ git checkout lclbranch
Switched to branch 'lclbranch'
Your branch is up to date with 'origin/lclbranch'.

$ git merge yzhbranch
Auto-merging GitStudy2.txt
hint: Waiting for your editor to close the file... unix2dos: converting file C:/Users/lcl/GitStudy/.g
it/MERGE_MSG to DOS format...
dos2unix: converting file C:/Users/lcl/GitStudy/.git/MERGE_MSG to Unix format...
Merge made by the 'ort' strategy.
 GitStudy2.txt | 4 ++++
 1 file changed, 4 insertions(+)

$ git push
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:Cailian-Liu/GitStudy.git
   9ce72dd..a1085cc  lclbranch -> lclbranch
```

Pull requests

Pull requests are not a feature of the git software itself, but of the online providers. They let a team discuss and review a commit before merging it into a shared branch such as develop or main. Depending on the provider, branches can also be protected or assigned owners so that only the branch owner or developers with the right permissions can commit on certain branches.

The procedure for merging with a pull request on github, which you should try out:

- Commit and push your feature branch.
- On github.com in your repository, choose *Pull Requests* in the top bar, then *New Pull Request*.
- Set the *base* branch as the one you want to merge into, e.g. develop, and the *compare* branch as the one with your changes. Select *Create Pull Request*.
- Add a title and description to start a discussion, then press *Create Pull Request* again to create the request.

Anyone in the team can now go to *Pull Requests* in the top bar of the repository page and see the open requests. You can either comment on them, or if it is your role in the team to approve the request for this branch, you can approve the pull request which creates a merge.

Since a pull request is linked to a branch, you can use it for code review as follows:

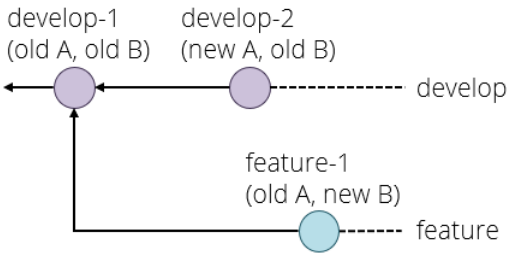
1. A developer creates a feature branch and submits a pull request.
2. The reviewer looks at the request. If they find bugs or other problems, they add a comment to the discussion.
3. The developer can address reviewer comments by making a new commit on their feature branch and pushing it, which automatically gets added to the discussion.
4. When the reviewer is happy, they approve the request which merges the latest version of the feature branch into the base branch (for example `develop`).

There is just one complication left. Suppose the following happens:

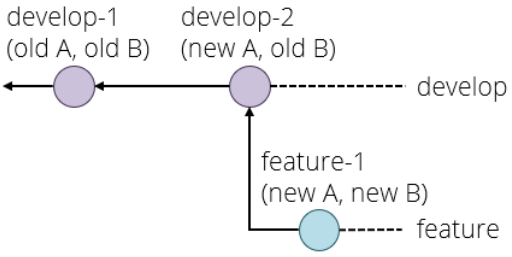
- Your project starts out with commit `develop-1` setting up the initial version of the develop branch. Imagine there are two files, A and B.

- You create a feature branch and make a commit `feature-1` which changes only file B.
- In the meantime, someone else makes a feature that changes file A, and merges it as `develop-2` to the develop branch.

You are now in the situation that `develop-2` has (new A, old B) and your `feature-1` has (old A, new B). Neither of these is what you want, you presumably want (new A, new B). We have met this situation before, but without branches. Graphically:



The solution here is to *rebase* your branch onto the latest commit on develop with `git rebase develop` and fix any conflicts that that causes, which produces the following situation:



If you now try and push your feature branch, you might get an error because the version of your feature branch on the origin repository still has the old version. The solution here is to force the push, which overwrites the old version, with

```
git push --force origin BRANCHNAME
```

This is a *think before you type* kind of command because it can break things for other developers if you do it on a shared branch. The basic safety rules are:

- Only rebase on *private* branches.
- Only force push on *private* branches, and only if it is absolutely necessary (for example to tidy up a rebase).

A private branch is one that you know no-one else is working on, for example your own feature branches.

If you ever get into a situation where you need to rebase or force push on a shared branch such as develop or main, you generally need to make sure that everyone on the team knows what is going on, synchronises their repositories both before and after the dangerous operation, and does not make any commits or pushes while someone is working on it - basically they need, in concurrency terms, an exclusive lock on the whole repository while doing this operation.

This is one reason why the main and develop branches are kept separate - and some workflows even include a third branch called `release`. If merges into main or release only ever come from the develop branch, then a situation where you need to rebase these branches can never happen.

To summarise, the pull request workflow is:

1. Commit and push your changes.
2. If necessary, rebase your feature branch on the develop branch.
3. Create a pull request.
4. If necessary, participate in a discussion or review and make extra commits to address points that other developers have raised.
5. Someone - usually not the developer who created the pull request - approves it, creating a merge commit in develop (or main).

w6_build

Build Tools

Part 1: Building C code from source

Extract it with `tar zxvf FILENAME`. The `z` means decompress it with `gunzip` first, `x` is for extract, `v` is for verbose (list the files it is expanding) and `f` is for read the archive from a file as opposed to standard input.

You can see a file called `INSTALL` which you can open in a text editor to find the standard instructions:

> Briefly, the shell commands ./configure; make; make install should configure, build, and install this package.

Installing

If you want to, you can now type `make install` to copy the executable to `/usr/local/bin`; but you'll probably find it fails with permissions issues. Turns out
most user's aren't allowed to install software for everyone.

- You could install using the `root` user who is allowed to install to the system directories: `sudo make install`
- You could install it somewhere else. Clean out your build and try rerunning `./configure` with `-prefix=${HOME}/.local/` first!

What to do if it doesn't build?

What do you do if it says it can't find a `.h` file, or can't link it to a library file (a `.so`)? C predates modern languages with package managers, so it probably means you haven't installed a library the code depends on. Luckily `apt-file` on Debian-based systems can be really helpful here: run `apt-file search <name of file>` to find out which package provides the file you're missing and install it.

Makefile

```
CC?=gcc
CFLAGS?=-Wall

.DEFAULT:
	echo "default"
rm:
	rm -rf driver library.o
driver: driver.c library.o
	${CC} ${CFLAGS} driver.c library.o
library.o: library.c
	${CC} ${CFLAGS} -c library.c -o library.o
install: driver
	install -m 0755 -o root -g root -s driver "$PREFIX/driver"
```

CC as a variable and shift to another compiler:

```
$ make CC=clang driver
```

Patternrule:

```
CC?=gcc
CFLAGS?=-Wall

.DEFAULT:
	echo "default"
%.o: %.c
	${CC} ${CFLAGS} -c -$^ -o $@
%: %.c %.o
	${CC} ${CFLAGS} -$^ -o $@

library.o: library.c
driver: driver.c library.o
```

Python

The Python programming language comes with a package manager called `pip`. Find the package that provides it and install it (**hint**: how did we find a missing library in the C build tools?). We are going to practice installing the mistletoe module, which renders markdown into HTML.

- In python, try the line `import mistletoe` and notice that you get `ModuleNotFoundError: No module named 'mistletoe'`.
- Quit python again (Control-d) and try `pip3 install --user mistletoe`. You should get a success message (and possibly a warning, explained below).
- Open python again and repeat `import mistletoe`. This produces no output, so the module was loaded.

```
$ python3
Python 3.11.2 (main, Mar 13 2023, 12:18:29) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import mistletoe
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'mistletoe'

$ sudo apt install python3-pip

$ pip3 install --user mistletoe
error: externally-managed-environment

× This environment is externally managed
╰─> To install Python packages system-wide, try apt install
    python3-xyz, where xyz is the package you are trying to
    install.

    If you wish to install a non-Debian-packaged Python package,
    create a virtual environment using python3 -m venv path/to/venv.
    Then use path/to/venv/bin/python and path/to/venv/bin/pip. Make
    sure you have python3-full installed.

    If you wish to install a non-Debian packaged Python application,
    it may be easiest to use pipx install xyz, which will manage a
    virtual environment for you. Make sure you have pipx installed.

    See /usr/share/doc/python3.11/README.venv for more information.

note: If you believe this is a mistake, please contact your Python installation or OS distribution
provider. You can override this, at the risk of breaking your Python installation or OS, by passin
g --break-system-packages.
hint: See PEP 668 for the detailed specification.

$ pip3 install --user mistletoe --break-system-packages
Collecting mistletoe
  Downloading mistletoe-1.4.0-py3-none-any.whl (51 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 51.3/51.3 kB 4.7 MB/s eta 0:00:00
Installing collected packages: mistletoe
  WARNING: The script mistletoe is installed in '/home/vagrant/.local/bin' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn
-script-location.
Successfully installed mistletoe-1.4.0

$ python3
Python 3.11.2 (main, Sep 14 2024, 03:00:30) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import mistletoe
>>>
```

Create a small sample markdown file as follows, called `hello.md` for example:

```
# Markdown Example

Markdown is a *markup* language.
```

Open python again and type the following. You need to indent the last line (four spaces is usual) and press ENTER twice at the end.

```
import mistletoe
with open('hello.md', 'r') as file:
    mistletoe.markdown(file)
```

This should print the markdown rendered to HTML, e.g.

```
<h1>Markdown Example</h1>\n<p>Markdown is a <em>markup</em> language.</p>
```

Sometimes you'll see things telling you to install a package with `sudo pip install` but don't do that! It will break things horribly eventually. You can use pip without sudo, by passing the `--user` option which installs packages into a folder in your home directory (`~/.local`) instead of in `/usr` which normally requires root permissions.

Python used to manage your OS should be run by the system designers; Python used for your dev work should be managed by you. And never the twain shall meet.

Avoiding sudo

Python comes with a mechanism called venv which lets you create a virtual python install that is owned by a user: you can alter the libraries in that without `sudo` and without fear of mucking up your host system. Read the docs and get used to using it—it'll save you a world of pain later!

```
vagrant@debian12:~$ sudo apt install python3-venv -y
vagrant@debian12:~$ python3 -m venv myenv
vagrant@debian12:~$ source myenv/bin/activate
(myenv) vagrant@debian12:~$ pip install --user mistletoe
ERROR: Can not perform a '--user' install. User site-packages are not visible in this virtualenv.
(myenv) vagrant@debian12:~$ pip install mistletoe
Collecting mistletoe
  Using cached mistletoe-1.4.0-py3-none-any.whl (51 kB)
Installing collected packages: mistletoe
Successfully installed mistletoe-1.4.0
```

- `pip freeze | tee requirements.txt` will list all the packages your using and what version they are and save them in a file called `requirements.txt`.
- `pip install -r requirements.txt` will install them again!

  ```
  (myenv) vagrant@debian12:~$ pip freeze
  mistletoe==1.4.0
  (myenv) vagrant@debian12:~$ pip freeze | tee requirements.txt
  mistletoe==1.4.0
  (myenv) vagrant@debian12:~$ pip install -r requirements.txt
  Requirement already satisfied: mistletoe==1.4.0 in ./myenv/lib/python3.11/site-packages (from -r r
  equirements.txt (line 1)) (1.4.0)
  ```

This makes it *super easy* to ensure that someone looking at your code has all the right dependencies without having to reel off a list of *go install these libraries* (and will make anyone whoever has to mark your code happy and more inclined to give you marks).

Java

In the Java world,

- The `javac` compiler turns source files (`.java`) into `.class` files;
- The `jar` tool packs class files into `.jar` files;
- The `java` command runs class files or jar files.

A Java Runtime Environment (JRE) contains only the `java` command, which is all you need to run java applications if you don't want to do any development. Many operating systems allow you to double-click jar files (at least ones containing a special file called a `manifest`) to run them in a JRE. A Java Development Kit (JDK) contains the `javac` and `jar` tools as well as a JRE. This is what you

need to develop in java. `maven` is a Java package manager and build tool. It is not part of the Java distribution, so you will need to install it separately.

Installing on Debian

On Debian, install the `openjdk-17-jdk` and `maven` packages. This should set things up so you're ready to go but if you have *multiple versions* of Java installed you may need to set the `JAVA_HOME` and `PATH` variables to point to your install.

For example:

```
export JAVA_HOME='/usr/lib/jvm/java-17-openjdk'export PATH="${PATH}:${JAVA_HOME}/bin"
```

|||advanced Debian also has a special command called `update-alternatives` that can help manage alternative development environments for you. Read the manual page! |||

```
vagrant@debian12:~$ sudo apt install openjdk-17-jdk
vagrant@debian12:~$ echo $JAVA_HOME

vagrant@debian12:~$ echo $PATH
/home/vagrant/.local/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games

vagrant@debian12:~$ java -version
openjdk version "17.0.13" 2024-10-15
OpenJDK Runtime Environment (build 17.0.13+11-Debian-2deb12u1)
OpenJDK 64-Bit Server VM (build 17.0.13+11-Debian-2deb12u1, mixed mode, sharing)

vagrant@debian12:~$ sudo update-alternatives --config java
There is 1 choice for the alternative java (providing /usr/bin/java).

  Selection    Path                                          Priority   Status
------------------------------------------------------------
* 0            /usr/lib/jvm/java-17-openjdk-amd64/bin/java   1711      auto mode
  1            /usr/lib/jvm/java-17-openjdk-amd64/bin/java   1711      manual mode

Press <enter> to keep the current choice[*], or type selection number: 0

vagrant@debian12:~$ export JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64/
vagrant@debian12:~$ export PATH=$JAVA_HOME/bin:$PATH
vagrant@debian12:~$ source ~/.bashrc
vagrant@debian12:~$ echo $JAVA_HOME
/usr/lib/jvm/java-17-openjdk-amd64/
vagrant@debian12:~$ echo $PATH
/usr/lib/jvm/java-17-openjdk-amd64//bin:/home/vagrant/.local/bin:/usr/local/bin:/usr/bin:/bin:/usr/lo
cal/games:/usr/games
vagrant@debian12:~$ sudo apt install maven
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
maven is already the newest version (3.8.7-1).
0 upgraded, 0 newly installed, 0 to remove and 77 not upgraded.
```

Running maven

Open a shell and type `mvn archetype:generate`. This lets you *generate an artifact from an archetype*, which is maven-speak for create a new folder with a maven file.

If you get a "not found" error, then most likely the maven `bin` folder is not on your path. If you're on a POSIX system and have used your package manager, this should be set up automatically, but if you've downloaded and unzipped maven then you have to `export PATH="$PATH:..."` where you replace the three dots with the path to the folder, and preferably put that line in your `~/.profile` too.

The first time you run it, maven will download a lot of libraries.

Maven will first show a list of all archetypes known to humankind (3046 at the time of counting) but you can just press ENTER to use the default, 2098 ("quickstart"). Maven now asks you for the version to use, press ENTER again.

You now have to enter the triple of (groupId, artifactId, version) for your project - it doesn't really matter but I suggest the following:

```
groupId: org.example
artifactId: project
version: 0.1
```

Just press ENTER again for the following questions, until you get a success message.

Maven has created a folder named after your artifactId, but you can move and rename it if you want and maven won't mind as long as you run it from inside the folder. Use `cd project` or whatever you called it to go inside the folder.

If you're in a POSIX shell, then `find .` should show everything in the folder (in Windows, `start .` opens it in Explorer instead):

```
.
./src
./src/main
./src/main/java
./src/main/java/org
./src/main/java/org/example
./src/main/java/org/example/App.java
./src/test
./src/test/java
./src/test/java/org
./src/test/java/org/example
./src/test/java/org/example/AppTest.java
./pom.xml
```

This is the standard maven folder structure. Your java sources live under `src/main/java`, and the default package name is `org.example` or whatever you put as your groupId so the main file is currently `src/main/java/org/example/App.java`. Since it's common to develop Java from inside an IDE or an editor with "folding" for paths (such as VS code), this folder structure is not a problem, although it's a bit clunky on the terminal.

The POM file

Have a look at `pom.xml` in an editor. The important parts you need to know about are:

The artifact's identifier (group id, artifact id, version):

```
<groupId>org.example</groupId>
<artifactId>project</artifactId>
<version>0.1</version>
```

The build properties determine what version of Java to compile against (by passing a flag to the compiler). Unfortunately, the default maven template seems to go with version 7, but version 8 was released back in 2014 which is stable enough for us, so please change the 1.7 to 1.8:

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

The dependencies section is where you add libraries you want to use. By default, your project uses `junit`, a unit testing framework - note that this is declared with `<scope>test</scope>` to say that it's only used for tests, not the project itself. You do not add this line when declaring your project's real dependencies.

```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

The `<plugins>` section contains the plugins that maven uses to compile and build your project. This section isn't mandatory, but it's included to "lock" the plugins to a particular version so that if a new version of a plugin is released, that doesn't change how your build works.

The one thing you should add here is the `exec-maven-plugin` as follows, so that you can actually run your project:

```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>exec-maven-plugin</artifactId>
    <version>3.0.0</version>
    <configuration>
        <mainClass>org.example.App</mainClass>
    </configuration>
</plugin>
```

The important line is the `mainClass` which you set to the full name (with path components) of your class with the `main()` function.

```
vagrant@debian12:~/project$ cat pom.xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instan
ce"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.example</groupId>
  <artifactId>project</artifactId>
  <version>0.1</version>

  <name>project</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.release>17</maven.compiler.release>
  </properties>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>org.junit</groupId>
        <artifactId>junit-bom</artifactId>
        <version>5.11.0</version>
        <type>pom</type>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>

  <dependencies>
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-api</artifactId>
      <scope>test</scope>
    </dependency>
    <!-- Optionally: parameterized tests support -->
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-params</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to
parent pom) -->
      <plugins>
        <!-- clean lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#cle
```

```
an_Lifecycle -->
        <plugin>
          <artifactId>maven-clean-plugin</artifactId>
          <version>3.4.0</version>
        </plugin>
        <!-- default lifecycle, jar packaging: see https://maven.apache.org/ref/current/maven-core/de
fault-bindings.html#Plugin_bindings_for_jar_packaging -->
        <plugin>
          <artifactId>maven-resources-plugin</artifactId>
          <version>3.3.1</version>
        </plugin>
        <plugin>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>3.13.0</version>
        </plugin>
        <plugin>
          <artifactId>maven-surefire-plugin</artifactId>
          <version>3.3.0</version>
        </plugin>
        <plugin>
          <artifactId>maven-jar-plugin</artifactId>
          <version>3.4.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-install-plugin</artifactId>
          <version>3.1.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-deploy-plugin</artifactId>
          <version>3.1.2</version>
        </plugin>
        <!-- site lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#site
_Lifecycle -->
        <plugin>
          <artifactId>maven-site-plugin</artifactId>
          <version>3.12.1</version>
        </plugin>
        <plugin>
          <artifactId>maven-project-info-reports-plugin</artifactId>
          <version>3.6.1</version>
        </plugin>
        <plugin>
          <groupId>org.codehaus.mojo</groupId>
          <artifactId>exec-maven-plugin</artifactId>
          <version>3.0.0</version>
          <configuration>
            <mainClass>org.example.App</mainClass>
          </configuration>
        </plugin>
      </plugins>
    </pluginManagement>
  </build>
</project>
```

Compile, run and develop

`mvn compile` compiles the project. The very first time you do this, it will download a lot of plugins, after that it will be pretty fast. Like `make`, it only compiles files that have changed since the last run, but if this ever gets out of sync (for example because you cancelled a compile halfway through) then `mvn clean` will remove all compiled files so the next compile will rebuild everything.

```
vagrant@debian12:~/project$ mvn compile
```

The `App.java` file contains a basic “Hello World!” program (have a look at this file). You can run the compiled project with `mvn exec:java` if you’ve set up the plugin as above. After you’ve run it the first time and it’s downloaded all the files it needs, lines coming from

maven itself will start with `[INFO]` or `[ERROR]` or similar, so lines without any prefix like that are printed by your program itself. You should see the hello world message on your screen.

```
vagrant@debian12:~/project$ mvn exec:java
vagrant@debian12:~/project$ mvn compile test exec:java
vagrant@debian12:~/project$ mvn test
vagrant@debian12:~/project$ mvn package
```

The development workflow is now as follows: you make your edits, then run `mvn compile test exec:java` to recompile, run your tests, then run the program. `mvn test` runs the tests in `src/test/java`. `mvn package` creates a jar file of your project in the `target/` folder.

I assume that you will be storing your Java projects in git repositories. In this case, you should create a file `.gitignore` in the same folder as the `pom.xml` and add the line `target/` to it, since you don't want the compiled classes and other temporary files and build reports in the repository. The `src/` folder, the `pom.xml` and the `.gitignore` file itself should all be checked in to the repository.

Adding a dependency

Thymeleaf is a Java templating library. It lets you write a template file or string for example

```
Hello, ${name}!
```

which you can later render with a particular name value. This is one of the standard ways of creating web applications, for example to display someone's profile page you would write a page template that takes care of the layout, styles, links etc. but uses template variables for the fields which you render when someone accesses the profile page for a particular person.

To use Thymeleaf or any other library, you first have to add it to your pom file. Go to mvnrepository.org and search for Thymeleaf, then find the latest stable ("release") version. There is a box where you can copy the `<dependency>` block to paste in your pom file. The next `mvn compile` will download thymeleaf and all its dependencies.

```
<!-- https://mvnrepository.com/artifact/org.thymeleaf/thymeleaf -->
<dependency>
    <groupId>org.thymeleaf</groupId>
    <artifactId>thymeleaf</artifactId>
    <version>3.1.2.RELEASE</version>
</dependency>
```

w7_debug

Debugging tools

strace - system calls

`strace -o tracelog FILE` save output of strace to tracelog

ltrace - library calls

strings- extract printable text strings

- `a` all sections. (default)
- `d or -data` only initialized and loaded data sections

getline

The `getline` function in C is used to read a line of input from a stream, such as `stdin` (standard input), into a dynamically allocated buffer. It is a safer alternative to functions like `gets` or `fgets` because it handles dynamic memory allocation and prevents buffer overflows.

Syntax

```
ssize_t getline(char **lineptr, size_t *n, FILE *stream);
```

1. `lineptr` **(char):****
   - A pointer to a `char *` (buffer) where the input will be stored.
   - If `lineptr` is `NULL`, `getline` will allocate memory automatically.
   - If `lineptr` is not `NULL`, it assumes the buffer already exists and resizes it if needed.
2. `n` **(size_t *):**
   - A pointer to a `size_t` variable that indicates the current size of the buffer pointed to by `lineptr`.
   - `getline` updates this value if it reallocates the buffer.
3. `stream` **(FILE *):**
   - The input stream to read from, such as `stdin` for standard input or a file pointer.

Return Value

- On Success:
  - Returns the number of characters read (including the newline `\n` but excluding the null terminator `\0`).
- On Failure:
  - Returns `1` if the end-of-file (EOF) is reached or an error occurs.
  - Sets `errno` to indicate the error.

gdb

`objdump -d FILE` display disassembly information

`run` run program (with arguments)

`bt` backtrace by reading stack

`b` create breakpoints

`d` delete breakpoints

`c` continue after hitting breakpoints

`p [variable]` print variable

`disas` disassembly

`info` information on registers or variables

`si` step into

`ni` next instruction

`x` examine memory

Command	Format
`x/s`	Display as a string (null-terminated).
`x/x`	Display as raw hexadecimal.
`x/d`	Display as a signed decimal.
`x/u`	Display as an unsigned decimal.
`x/t`	Display as binary.
`x/i`	Display as machine instructions (disassemble).

x86-64 assembly

registers:

`%rip` program counter

`%rax` Accumulator register, often used for storing return values or temporary data. (TEMP)

`%eax` Lower 32 bits of `%rax`. Contains the result of operations or comparisons.

`%al` Lower 8 bits of `%rax`. Used for byte-level operations (like comparing a single character).

`%rcx, %rdx, %rsi, %rdi` General-purpose registers for passing arguments or intermediate data. (ARG)

`%esi` Lower 32 bits of `%rsi`.

`%rbp` Base pointer, often used to reference local variables within the current stack frame. (LCL)

`%rsp` Stack pointer, points to the top of the stack (used for function calls and local storage). (SP)

64-bit Register	Lower 32 bits	Lower 16 bits	Lower 8 bits (High/Low)
`%rax`	`%eax`	`%ax`	`%al` (low), `%ah` (high)
`%rbx`	`%ebx`	`%bx`	`%bl` (low), `%bh` (high)
`%rcx`	`%ecx`	`%cx`	`%cl` (low), `%ch` (high)
`%rdx`	`%edx`	`%dx`	`%dl` (low), `%dh` (high)
`%rsi`	`%esi`	`%si`	`%sil` (low)
`%rdi`	`%edi`	`%di`	`%dil` (low)
`%rsp`	`%esp`	`%sp`	`%spl` (low)
`%rbp`	`%ebp`	`%bp`	`%bpl` (low)
`%r8` – `%r15`	`%r8d` – `%r15d`	`%r8w` – `%r15w`	`%r8b` – `%r15b`

`(%rax)` indirect addressing.

`4(%rax)` indirect addressing with offset 4 bytes

`$0x42` immediate value (constant)

1. Zero-Argument Instructions

These instructions operate without explicit operands. They often work with implicit registers or affect processor flags.

Instruction	Description
`ret`	Return from a function (pop address from stack).
`leave`	Restore the stack frame by resetting `%rbp` and `%rsp`.
`nop`	No operation (does nothing, often used for padding).
`pushf`	Push the processor flags register onto the stack.
`popf`	Pop the top of the stack into the processor flags register.
`clc`	Clear the carry flag (used in arithmetic).
`stc`	Set the carry flag.
`hlt`	Halt the CPU until the next interrupt.

2. One-Argument Instructions

These instructions operate on a single operand, which can be a register, memory address, or immediate value. The operation is often implicit (e.g., incrementing or negating the operand).

Instruction	Description
`callq <addr>`	Calls a function at `<addr>` (pushes return address onto the stack).

`push <src>`	Push the value in `<src>` onto the stack.
`pop <dst>`	Pop the top of the stack into `<dst>`.
`inc <dst>`	Increment the value in `<dst>` by 1.
`dec <dst>`	Decrement the value in `<dst>` by 1.
`neg <dst>`	Negate the value in `<dst>` (two's complement).
`not <dst>`	Perform a bitwise NOT on `<dst>` (flip all bits).
`jmp <addr>`	Unconditionally jump to `<addr>`.
`call <addr>`	Call a function at `<addr>` (push return address onto stack).
`test <dst>`	Perform a logical AND between `<dst>` and `<dst>`, setting flags but discarding the result.
`setcc <dst>`	Set the byte in `<dst>` based on a condition (`cc` specifies the condition, e.g., `sete` for equal).

3. Two-Argument Instructions

These are the most common instructions in x86-64. They perform an operation on the **source** operand (`<src>`) and store the result in the **destination** operand (`<dst>`). The operands can be registers, memory addresses, or immediate values.

Instruction	Description
`mov <src>, <dst>`	Move the value from `<src>` to `<dst>`.
`add <src>, <dst>`	Add `<src>` to `<dst>` and store the result in `<dst>`.
`sub <src>, <dst>`	Subtract `<src>` from `<dst>` and store the result in `<dst>`.
`cmp <src>, <dst>`	Compare `<src>` with `<dst>` (sets flags, no result stored).
`and <src>, <dst>`	Perform a bitwise AND between `<src>` and `<dst>`, storing the result in `<dst>`.
`or <src>, <dst>`	Perform a bitwise OR between `<src>` and `<dst>`, storing the result in `<dst>`.
`xor <src>, <dst>`	Perform a bitwise XOR between `<src>` and `<dst>`, storing the result in `<dst>`.
`lea <src>, <dst>`	Load the effective address of `<src>` into `<dst>` (used for pointer arithmetic).
`imul <src>, <dst>`	Multiply `<src>` with `<dst>` and store the result in `<dst>`.
`shr <src>, <dst>`	Shift `<dst>` right by `<src>` bits, filling with zeroes.
`shl <src>, <dst>`	Shift `<dst>` left by `<src>` bits, filling with zeroes.
`movzbl <src>, <dst>`	Move the zero-extended byte from `<src>` to `<dst>` (used to load 8-bit values into larger registers).

Cracks

```
$ strings -d ./crackme-1
...
strcmp
...
Beetlejuice
...

$ ltrace ./crackme-1
puts("What is the password?"What is the password?
)                                                = 22
getline(0x7fffbd391880, 0x7fffbd391878, 0x7f00de2449c0, 0x7fffbd391878
)   = 1
strcmp("", "Beetlejuice")                                                  = -66
puts("Nope"Nope
)                                                                 = 5
free(0x159a6b0)                                                            = <void>
+++ exited (status 1) +++
```

strcmp gives the password.

```
$ gdb ./crackme-2
(gdb) b main
(gdb) run <<< "abc"
(gdb) disas
Dump of assembler code for function main:
   0x0000000000400656 <+0>:     push   %rbp
```

```
   0x0000000000400657 <+1>:     mov    %rsp,%rbp
   0x000000000040065a <+4>:     sub    $0x20,%rsp
=> 0x000000000040065e <+8>:     movq   $0x0,-0x18(%rbp)
   0x0000000000400666 <+16>:    movq   $0x0,-0x20(%rbp)
   0x000000000040066e <+24>:    movq   $0x0,-0x10(%rbp)
   0x0000000000400676 <+32>:    mov    $0x400848,%edi
   0x000000000040067b <+37>:    callq  0x400550 <puts@plt>
   0x0000000000400680 <+42>:    mov    0x2009b9(%rip),%rdx        # 0x601040 <stdin@@GLIBC_2.2.5>
   0x0000000000400687 <+49>:    lea    -0x20(%rbp),%rcx
   0x000000000040068b <+53>:    lea    -0x18(%rbp),%rax
   0x000000000040068f <+57>:    mov    %rcx,%rsi
   0x0000000000400692 <+60>:    mov    %rax,%rdi
   0x0000000000400695 <+63>:    callq  0x400560 <getline@plt>
   0x000000000040069a <+68>:    mov    %rax,-0x10(%rbp)
   0x000000000040069e <+72>:    mov    -0x18(%rbp),%rax
   0x00000000004006a2 <+76>:    mov    -0x10(%rbp),%rdx
   0x00000000004006a6 <+80>:    sub    $0x1,%rdx
   0x00000000004006aa <+84>:    add    %rdx,%rax
   0x00000000004006ad <+87>:    movb   $0x0,(%rax)
   0x00000000004006b0 <+90>:    cmpq   $0xb,-0x10(%rbp)
   0x00000000004006b5 <+95>:    jne    0x40077b <main+293>
   0x00000000004006bb <+101>:   mov    -0x18(%rbp),%rax
   0x00000000004006bf <+105>:   movzbl (%rax),%eax
   0x00000000004006c2 <+108>:   cmp    $0x42,%al
   0x00000000004006c4 <+110>:   jne    0x40077b <main+293>
   0x00000000004006ca <+116>:   mov    -0x18(%rbp),%rax
   0x00000000004006ce <+120>:   add    $0x2,%rax
   0x00000000004006d2 <+124>:   movzbl (%rax),%eax
   0x00000000004006d5 <+127>:   cmp    $0x74,%al
   0x00000000004006d7 <+129>:   jne    0x40077b <main+293>
   0x00000000004006dd <+135>:   mov    -0x18(%rbp),%rax
   0x00000000004006e1 <+139>:   add    $0x8,%rax
   0x00000000004006e5 <+143>:   movzbl (%rax),%eax
   0x00000000004006e8 <+146>:   cmp    $0x73,%al
   0x00000000004006ea <+148>:   jne    0x40077b <main+293>
   0x00000000004006f0 <+154>:   mov    -0x18(%rbp),%rax
   0x00000000004006f4 <+158>:   add    $0x3,%rax
   0x00000000004006f8 <+162>:   movzbl (%rax),%eax
   0x00000000004006fb <+165>:   cmp    $0x65,%al
   0x00000000004006fd <+167>:   jne    0x40077b <main+293>
   0x00000000004006ff <+169>:   mov    -0x18(%rbp),%rax
   0x0000000000400703 <+173>:   add    $0x9,%rax
   0x0000000000400707 <+177>:   movzbl (%rax),%eax
   0x000000000040070a <+180>:   cmp    $0x65,%al
   0x000000000040070c <+182>:   jne    0x40077b <main+293>
   0x000000000040070e <+184>:   mov    -0x18(%rbp),%rax
   0x0000000000400712 <+188>:   add    $0x6,%rax
   0x0000000000400716 <+192>:   movzbl (%rax),%eax
   0x0000000000400719 <+195>:   cmp    $0x65,%al
   0x000000000040071b <+197>:   jne    0x40077b <main+293>
   0x000000000040071d <+199>:   mov    -0x18(%rbp),%rax
   0x0000000000400721 <+203>:   add    $0x1,%rax
   0x0000000000400725 <+207>:   movzbl (%rax),%eax
   0x0000000000400728 <+210>:   cmp    $0x65,%al
   0x000000000040072a <+212>:   jne    0x40077b <main+293>
   0x000000000040072c <+214>:   mov    -0x18(%rbp),%rax
   0x0000000000400730 <+218>:   add    $0x5,%rax
   0x0000000000400734 <+222>:   movzbl (%rax),%eax
   0x0000000000400737 <+225>:   cmp    $0x67,%al
   0x0000000000400739 <+227>:   jne    0x40077b <main+293>
   0x000000000040073b <+229>:   mov    -0x18(%rbp),%rax
   0x000000000040073f <+233>:   add    $0x4,%rax
   0x0000000000400743 <+237>:   movzbl (%rax),%eax
   0x0000000000400746 <+240>:   cmp    $0x6c,%al
```

```
   0x0000000000400748 <+242>:   jne    0x40077b <main+293>
   0x000000000040074a <+244>:   mov    -0x18(%rbp),%rax
   0x000000000040074e <+248>:   add    $0x7,%rax
   0x0000000000400752 <+252>:   movzbl (%rax),%eax
   0x0000000000400755 <+255>:   cmp    $0x75,%al
   0x0000000000400757 <+257>:   jne    0x40077b <main+293>
   0x0000000000400759 <+259>:   mov    -0x18(%rbp),%rax
   0x000000000040075d <+263>:   add    $0xa,%rax
   0x0000000000400761 <+267>:   movzbl (%rax),%eax
   0x0000000000400764 <+270>:   test   %al,%al
   0x0000000000400766 <+272>:   jne    0x40077b <main+293>
   0x0000000000400768 <+274>:   mov    $0x40085e,%edi
   0x000000000040076d <+279>:   callq  0x400550 <puts@plt>
   0x0000000000400772 <+284>:   movl   $0x0,-0x4(%rbp)
   0x0000000000400779 <+291>:   jmp    0x40078c <main+310>
   0x000000000040077b <+293>:   mov    $0x400867,%edi
   0x0000000000400780 <+298>:   callq  0x400550 <puts@plt>
   0x0000000000400785 <+303>:   movl   $0x1,-0x4(%rbp)
   0x000000000040078c <+310>:   mov    -0x18(%rbp),%rax
   0x0000000000400790 <+314>:   test   %rax,%rax
   0x0000000000400793 <+317>:   je     0x4007a1 <main+331>
   0x0000000000400795 <+319>:   mov    -0x18(%rbp),%rax
   0x0000000000400799 <+323>:   mov    %rax,%rdi
   0x000000000040079c <+326>:   callq  0x400540 <free@plt>
   0x00000000004007a1 <+331>:   mov    -0x4(%rbp),%eax
   0x00000000004007a4 <+334>:   leaveq
   0x00000000004007a5 <+335>:   retq
End of assembler dump.
```

look up all the *cmp*s:

```
$ objdump -d ./crackme-2 | grep -w cmp
  4005be:       48 39 f8                cmp    %rdi,%rax
  4006c2:       3c 42                   cmp    $0x42,%al
  4006d5:       3c 74                   cmp    $0x74,%al
  4006e8:       3c 73                   cmp    $0x73,%al
  4006fb:       3c 65                   cmp    $0x65,%al
  40070a:       3c 65                   cmp    $0x65,%al
  400719:       3c 65                   cmp    $0x65,%al
  400728:       3c 65                   cmp    $0x65,%al
  400737:       3c 67                   cmp    $0x67,%al
  400746:       3c 6c                   cmp    $0x6c,%al
  400755:       3c 75                   cmp    $0x75,%al
  400801:       48 39 dd                cmp    %rbx,%rbp
```

the first arguments form the password in order.

```
$ gdb ./crackme-3
(gdb) b main
(gdb) run <<< "abc"
(gdb) disas
Dump of assembler code for function main:
   0x00000000004006f6 <+0>:     push   %rbp
   0x00000000004006f7 <+1>:     mov    %rsp,%rbp
   0x00000000004006fa <+4>:     sub    $0x20,%rsp
=> 0x00000000004006fe <+8>:     movq   $0x0,-0x18(%rbp)
   0x0000000000400706 <+16>:    movq   $0x0,-0x20(%rbp)
   0x000000000040070e <+24>:    movl   $0x0,-0xc(%rbp)
   0x0000000000400715 <+31>:    mov    $0x400898,%edi
   0x000000000040071a <+36>:    callq  0x4005d0 <puts@plt>
   0x000000000040071f <+41>:    mov    0x20092a(%rip),%rdx        # 0x601050 <stdin@@GLIBC_2.2.5>
   0x0000000000400726 <+48>:    lea    -0x20(%rbp),%rcx
   0x000000000040072a <+52>:    lea    -0x18(%rbp),%rax
   0x000000000040072e <+56>:    mov    %rcx,%rsi
```

```
   0x0000000000400731 <+59>:    mov    %rax,%rdi
   0x0000000000400734 <+62>:    callq  0x400600 <getline@plt>
   0x0000000000400739 <+67>:    mov    %eax,-0xc(%rbp)
   0x000000000040073c <+70>:    cmpl   $0x0,-0xc(%rbp)
   0x0000000000400740 <+74>:    jns    0x40074c <main+86>
   0x0000000000400742 <+76>:    mov    $0x1,%eax
   0x0000000000400747 <+81>:    jmpq   0x4007fd <main+263>
   0x000000000040074c <+86>:    mov    -0x18(%rbp),%rax
   0x0000000000400750 <+90>:    mov    -0xc(%rbp),%edx
   0x0000000000400753 <+93>:    movslq %edx,%rdx
   0x0000000000400756 <+96>:    sub    $0x1,%rdx
   0x000000000040075a <+100>:   add    %rdx,%rax
   0x000000000040075d <+103>:   movb   $0x0,(%rax)
   0x0000000000400760 <+106>:   movl   $0x0,-0x8(%rbp)
   0x0000000000400767 <+113>:   jmp    0x40078f <main+153>
   0x0000000000400769 <+115>:   mov    -0x18(%rbp),%rdx
   0x000000000040076d <+119>:   mov    -0x8(%rbp),%eax
   0x0000000000400770 <+122>:   cltq
   0x0000000000400772 <+124>:   add    %rdx,%rax
   0x0000000000400775 <+127>:   movzbl (%rax),%ecx
   0x0000000000400778 <+130>:   mov    -0x18(%rbp),%rdx
   0x000000000040077c <+134>:   mov    -0x8(%rbp),%eax
   0x000000000040077f <+137>:   cltq
   0x0000000000400781 <+139>:   add    %rdx,%rax
   0x0000000000400784 <+142>:   xor    $0x42,%ecx
   0x0000000000400787 <+145>:   mov    %ecx,%edx
   0x0000000000400789 <+147>:   mov    %dl,(%rax)
   0x000000000040078b <+149>:   addl   $0x1,-0x8(%rbp)
   0x000000000040078f <+153>:   mov    -0xc(%rbp),%eax
   0x0000000000400792 <+156>:   sub    $0x1,%eax
   0x0000000000400795 <+159>:   cmp    %eax,-0x8(%rbp)
   0x0000000000400798 <+162>:   jl     0x400769 <main+115>
   0x000000000040079a <+164>:   mov    -0x18(%rbp),%rax
   0x000000000040079e <+168>:   mov    %rax,%rdi
   0x00000000004007a1 <+171>:   callq  0x4005e0 <strlen@plt>
   0x00000000004007a6 <+176>:   cmp    $0xa,%rax
   0x00000000004007aa <+180>:   jne    0x4007d4 <main+222>
   0x00000000004007ac <+182>:   mov    -0x18(%rbp),%rax
   0x00000000004007b0 <+186>:   mov    $0x4008ae,%esi
   0x00000000004007b5 <+191>:   mov    %rax,%rdi
   0x00000000004007b8 <+194>:   callq  0x4005f0 <strcmp@plt>
   0x00000000004007bd <+199>:   test   %eax,%eax
   0x00000000004007bf <+201>:   jne    0x4007d4 <main+222>
   0x00000000004007c1 <+203>:   mov    $0x4008b9,%edi
   0x00000000004007c6 <+208>:   callq  0x4005d0 <puts@plt>
   0x00000000004007cb <+213>:   movl   $0x0,-0x4(%rbp)
   0x00000000004007d2 <+220>:   jmp    0x4007e5 <main+239>
   0x00000000004007d4 <+222>:   mov    $0x4008c2,%edi
   0x00000000004007d9 <+227>:   callq  0x4005d0 <puts@plt>
   0x00000000004007de <+232>:   movl   $0x1,-0x4(%rbp)
   0x00000000004007e5 <+239>:   mov    -0x18(%rbp),%rax
   0x00000000004007e9 <+243>:   test   %rax,%rax
   0x00000000004007ec <+246>:   je     0x4007fa <main+260>
   0x00000000004007ee <+248>:   mov    -0x18(%rbp),%rax
   0x00000000004007f2 <+252>:   mov    %rax,%rdi
   0x00000000004007f5 <+255>:   callq  0x4005c0 <free@plt>
   0x00000000004007fa <+260>:   mov    -0x4(%rbp),%eax
   0x00000000004007fd <+263>:   leaveq
   0x00000000004007fe <+264>:   retq
End of assembler dump.
```

from `main+115` to `main+162` is a loop iterating through the input string and XORs each character with `0x42`. strcmp is called at main+194 callq (0x4005f0 is where strcmp is stored at). strcmp always compares `%rsi` and `%rdi`, so we look up `main+191` and `main+186`, and found an address 0x4008ae. type `x/s 0x4008ae` returns

```
(gdb) x/s 0x4008ae
0x4008ae:       "\017'\021#;\006#;r*"
```

which contains escaped octal code (since they are non-printable characters). revert this string with xor 0x42.

```
$ ltrace ./crackme-4
puts("What is the password?"What is the password?
)                                               = 22
getline(0x7ffc7d25b0c8, 0x7ffc7d25b0c0, 0x7f9c26e169c0, 0x7ffc7d25b0c0
)   = 1
srand(0)                                                                  = <void>
rand()                                                                    = 1804289383
atoi(0xe7a6b0, 0x7ffc7d25b094, 0, 0x7f9c26e161e8)                         = 0
puts("Nope"Nope
)                                                               = 5
free(0xe7a6b0)                                                            = <void>
+++ exited (status 1) +++

$ gdb ./crackme-4
(gdb) b main
(gdb) run <<< "password"
(gdb) disas
Dump of assembler code for function main:
   0x0000000000400736 <+0>:     push   %rbp
   0x0000000000400737 <+1>:     mov    %rsp,%rbp
   0x000000000040073a <+4>:     sub    $0x20,%rsp
=> 0x000000000040073e <+8>:     movq   $0x0,-0x18(%rbp)
   0x0000000000400746 <+16>:    movq   $0x0,-0x20(%rbp)
   0x000000000040074e <+24>:    movq   $0x0,-0x10(%rbp)
   0x0000000000400756 <+32>:    mov    $0x400898,%edi
   0x000000000040075b <+37>:    callq  0x400600 <puts@plt>
   0x0000000000400760 <+42>:    mov    0x2008e9(%rip),%rdx        # 0x601050 <stdin@@GLIBC_2.2.5>
   0x0000000000400767 <+49>:    lea    -0x20(%rbp),%rcx
   0x000000000040076b <+53>:    lea    -0x18(%rbp),%rax
   0x000000000040076f <+57>:    mov    %rcx,%rsi
   0x0000000000400772 <+60>:    mov    %rax,%rdi
   0x0000000000400775 <+63>:    callq  0x400630 <getline@plt>
   0x000000000040077a <+68>:    mov    %rax,-0x10(%rbp)
   0x000000000040077e <+72>:    mov    -0x18(%rbp),%rax
   0x0000000000400782 <+76>:    mov    -0x10(%rbp),%rdx
   0x0000000000400786 <+80>:    sub    $0x1,%rdx
   0x000000000040078a <+84>:    add    %rdx,%rax
   0x000000000040078d <+87>:    movb   $0x0,(%rax)
   0x0000000000400790 <+90>:    mov    $0x0,%edi
   0x0000000000400795 <+95>:    callq  0x400610 <srand@plt>
   0x000000000040079a <+100>:   callq  0x400640 <rand@plt>
   0x000000000040079f <+105>:   mov    %eax,-0x4(%rbp)
   0x00000000004007a2 <+108>:   mov    -0x18(%rbp),%rax
   0x00000000004007a6 <+112>:   mov    %rax,%rdi
   0x00000000004007a9 <+115>:   callq  0x400620 <atoi@plt>
   0x00000000004007ae <+120>:   cmp    %eax,-0x4(%rbp)
   0x00000000004007b1 <+123>:   jne    0x4007c6 <main+144>
   0x00000000004007b3 <+125>:   mov    $0x4008ae,%edi
   0x00000000004007b8 <+130>:   callq  0x400600 <puts@plt>
   0x00000000004007bd <+135>:   movl   $0x0,-0x4(%rbp)
   0x00000000004007c4 <+142>:   jmp    0x4007d7 <main+161>
   0x00000000004007c6 <+144>:   mov    $0x4008b7,%edi
   0x00000000004007cb <+149>:   callq  0x400600 <puts@plt>
   0x00000000004007d0 <+154>:   movl   $0x1,-0x4(%rbp)
   0x00000000004007d7 <+161>:   mov    -0x18(%rbp),%rax
   0x00000000004007db <+165>:   test   %rax,%rax
   0x00000000004007de <+168>:   je     0x4007ec <main+182>
   0x00000000004007e0 <+170>:   mov    -0x18(%rbp),%rax
   0x00000000004007e4 <+174>:   mov    %rax,%rdi
```

```
   0x00000000004007e7 <+177>:   callq  0x4005f0 <free@plt>
   0x00000000004007ec <+182>:   mov    -0x4(%rbp),%eax
   0x00000000004007ef <+185>:   leaveq
   0x00000000004007f0 <+186>:   retq
End of assembler dump.
```

combining `ltrace` with `gdb` we know at `main+95` the seed is set to `srand(0)`. there is only one `cmp`, namely `main+120`, and its second argument is `-0x4(%rbp)`. so we go all the back to find this address until `main+105`, where the value of `%eax` is sent to this address. and right above it is `callq` `rand`, so the `cmp` compares `%eax` from `main+115`, `callq` `atoi` with `rand`. and from `ltrace` we know the output of `rand` is `1804289383`. Alternative approach: extract the value of that address:

```
(gdb) b *0x4007ae
(gdb) run <<< "a"
(gdb) info registers rbp
rbp            0x7fffffffd3b0      0x7fffffffd3b0
(gdb) x/wx 0x7fffffffd3ac
0x7fffffffd3ac: 0x6b8b4567
(gdb) x/u 0x7fffffffd3ac
0x7fffffffd3ac: 1804289383
(gdb) x/d 0x7fffffffd3ac
0x7fffffffd3ac: 1804289383
```

also reveals the password.

```
$ strings -d ./crackme-5
...
strncmp
puts
tolower
getchar
toupper
...
B3t3Lg3uS3
...

$ ltrace ./crackme-5
puts("What is the password?"What is the password?
)                                               = 22
getchar(0, 0x25472a0, 0x7f0c39975860, 0x7f0c396d15a8
)                     = 10
strncmp("", "B3t3Lg3uS3", 10)                                                 = -66
puts("Nope"Nope
)                                                                 = 5
+++ exited (status 0) +++

$ strace ./crackme-5
execve("./crackme-5", ["./crackme-5"], 0x7fffc29fb9a0 /* 49 vars */) = 0
...
write(1, "What is the password?\n", 22What is the password?
) = 22
...
read(0,
"\n", 1024)                     = 1
write(1, "Nope\n", 5Nope
)                   = 5
exit_group(0)                           = ?
+++ exited with 0 +++

$ gdb ./crackme-5
(gdb) b main
(gdb) run <<< ""
(gdb) disas
Dump of assembler code for function main:
   0x0000000000400746 <+0>:     push   %rbp
```

```
   0x0000000000400747 <+1>:     mov    %rsp,%rbp
   0x000000000040074a <+4>:     sub    $0x10,%rsp
=> 0x000000000040074e <+8>:     mov    $0x400a78,%edi
   0x0000000000400753 <+13>:    callq  0x400620 <puts@plt>
   0x0000000000400758 <+18>:    jmp    0x400765 <main+31>
   0x000000000040075a <+20>:    movsbl -0x1(%rbp),%eax
   0x000000000040075e <+24>:    mov    %eax,%edi
   0x0000000000400760 <+26>:    callq  0x4007c1 <interpret>
   0x0000000000400765 <+31>:    callq  0x400630 <getchar@plt>
   0x000000000040076a <+36>:    mov    %al,-0x1(%rbp)
   0x000000000040076d <+39>:    cmpb   $0xff,-0x1(%rbp)
   0x0000000000400771 <+43>:    je     0x400779 <main+51>
   0x0000000000400773 <+45>:    cmpb   $0xa,-0x1(%rbp)
   0x0000000000400777 <+49>:    jne    0x40075a <main+20>
   0x0000000000400779 <+51>:    mov    $0xa,%edx
   0x000000000040077e <+56>:    mov    $0x400a8e,%esi
   0x0000000000400783 <+61>:    mov    $0x602070,%edi
   0x0000000000400788 <+66>:    callq  0x400600 <strncmp@plt>
   0x000000000040078d <+71>:    test   %eax,%eax
   0x000000000040078f <+73>:    jne    0x4007ab <main+101>
   0x0000000000400791 <+75>:    movzbl 0x2018e8(%rip),%eax        # 0x602080 <failed>
   0x0000000000400798 <+82>:    xor    $0x1,%eax
   0x000000000040079b <+85>:    test   %al,%al
   0x000000000040079d <+87>:    je     0x4007ab <main+101>
   0x000000000040079f <+89>:    mov    $0x400a99,%edi
   0x00000000004007a4 <+94>:    callq  0x400620 <puts@plt>
   0x00000000004007a9 <+99>:    jmp    0x4007b5 <main+111>
   0x00000000004007ab <+101>:   mov    $0x400aa2,%edi
   0x00000000004007b0 <+106>:   callq  0x400620 <puts@plt>
   0x00000000004007b5 <+111>:   movzbl 0x2018c4(%rip),%eax        # 0x602080 <failed>
   0x00000000004007bc <+118>:   movzbl %al,%eax
   0x00000000004007bf <+121>:   leaveq
   0x00000000004007c0 <+122>:   retq
End of assembler dump.
```

from disas we know there is an `interpreter` function altering the input string. we can use the following to determine which characters are strange:

```
$ ltrace ./crackme-5 <<< "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
$ ltrace ./crackme-5 <<< "abcdefghijklmnopqrstuvwxyz"
$ ltrace ./crackme-5 <<< "\!\@\#\$\%\^\&\*\(\)\-\_\+\=\{\}\[\]\|\\\:\;\"\'\<\>\,\.\?\/"
$ ltrace ./crackme-5 <<< "1234567890"
```

and can directly observe from the output that `c, d, l, U, +, -, >, <, .` are distinct and likely to correspond to commands. next step is go back to `gdb` and repeat the following process:

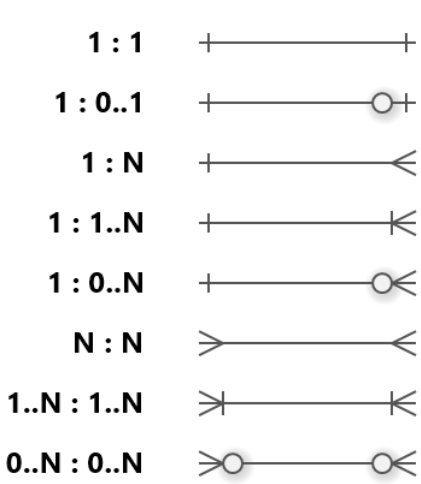
1. set breakpoint at interpret then run with the distinct chars.

   ```
   b interpret
   run <<< "++c>>d--l<<U."
   ```

2. use `si` to step next, use `p i` to observe the value of loop variable `i`, use `p command` to remind the current char being examined (current char of the input string), use `x/s memory` view the current value of memory, in order to deduce the utility of that char.
3. after figuring out the effect of every char, the puzzle is solved.

w8+w9_sql

Entity-Relationship Diagram



crow's foot notation

SQLite Query

Elections

1. List the names of all parties that stood in the election, ordered alphabetically by name.

```
select name from party order by party.name asc;
```

2. List the names of all parties that stood in the Bedminster ward.

```
select party.name from party where party.id in ( select candidate.party from candidate join ward o
n candidate.ward = ( select ward.id where ward.name = 'Bedminster' ) );
```

3. How many votes did Labour get in the Stockwood ward?

```
select sum(c.votes) from candidate c join party p on p.id = c.party join ward w on w.id = c.ward w
here p.name='Labour' and w.name='Stockwood';
```

4. List the names, parties and number of votes obtained for all candidates in the Southville ward. Order the candidates by number of votes obtained descending (winner comes first).

```
select c.name, p.name, c.votes from candidate c join party p on p.id = c.party join ward w on w.id
=c.ward where w.name = 'Southville' order by c.votes desc;
```

5. List the name, party and number of votes obtained for the winner only in the Knowle ward.

```
select c.name, p.name, c.votes from candidate c join party p on p.id = c.party join ward w on w.id
=c.ward where w.name = 'Knowle' order by c.votes desc limit 1;
```

6. How many votes were cast in all of Bristol in the 2014 elections?

```
select sum(c.votes) from candidate c;
```

7. How many votes were cast in the 'Windmill Hill' ward and what percentage of the electorate in this ward does this represent? Your statement should produce a table with one row and two columns called 'votes' and 'percentage'.

```
select sum(c.votes) as votes, (sum(c.votes)*100.0/w.electorate) as percentage from candidate c joi
n ward w on w.id = c.ward where w.name='Windmill Hill';
```

8. List the names, parties and percentage of votes obtained for all candidates in the Southville ward. Order the candidates by percentage of votes obtained descending.

```
select c.name, p.name, c.votes, sum(c.votes)*100.0/w.electorate as percentage from candidate c joi
n party p on p.id = c.party join ward w on w.id=c.ward where w.name = 'Southville' order by percen
tage desc;
```

9. How successful (in % of votes cast) was the Conservative party in each ward?

```
select w.name, c.votes*100.0/w.electorate as per from candidate c join ( select ward, sum(candidat
e.votes) total_votes from candidate group by ward) as wv on w.id = wv.ward join party p on p.id =
c.party join ward w on w.id=c.ward where p.name = 'Conservative' group by w.name, wv.total_votes o
rder by per;
```

10. Which rank did Labour end up in the 'Whitchurch Park' ward? Your statement should produce a table with a single row and column containing the answer as a number. You can assume no ties.

```
select ranking from (select c.votes, p.name pname, rank() over (order by c.votes desc) ranking fro
m candidate c join ward w on w.id=c.ward join party p on p.id=c.party where w.name='Whitchurch Par
k') as ranked where pname='Labour';
```

11. What is the total number of votes that each party got in the elections? Your result should be a table with two columns party, votes.

```
select p.name, sum(c.votes) votes from candidate c join party p on p.id=c.party group by p.name or
der by votes desc;
```

12. Find all wards where the Green party beat Labour and create a table with two columns ward, difference where the difference column is the number of Green votes minus the number of Labour votes. Your table should be ordered by difference, with the highest one first.

```
select green.ward, green.votes-labour.votes diff
from
(select w.name ward, c.votes votes from candidate c join party p on p.id=c.party join ward w on w.
id=c.ward where p.name='Green') green
join
(select w.name ward, c.votes votes from candidate c join party p on p.id=c.party join ward w on w.
id=c.ward where p.name='Labour') labour
on green.ward=labour.ward
where green.votes>=labour.votes
order by diff desc;
```

w10_datalog

Week 10: Datalog

The Addams Family

Task 1. Capture the Addams Family tree using Horn clauses. To test run the query `parent(X,sharron)` to see if *Sharron* is anyone's parent. The only match should be *Debbie*.

Task 2. A sibling is someone who shares a parent with you. Define a rule `sibling(X,Y)` that captures this. Check that Gomez's sibling is Fester.

Task 3. AbcDatalog allows negation in rules. We can specify that two variables are not equal in a rule by saying `X!=Y`. Check whether `sibling(wednesday,wednesday)` is true, and if so fix it.

Task 4. A cousin is someone you share a grandparent with, but who isn't your sibling. Write the rule `cousin(X,Y)` and check that Gomez and Morticia are cousins.

Task 5. An aunt or uncle is your parent's sibling. Define the rule and query whose Uncle Fester is.

Task 6. Who are Pugsley's aunts and uncles? What about Debbie?

```
% Parent relationships
parent(debbie, sharron). parent(debbie, dave). parent(fester, eudora).
parent(fester, father). parent(gomez, eudora). parent(gomez, father).
...

% rules
grandparent(X,Z) :- parent(X,Y), parent(Y,Z).
sibling(X,Y) :- parent(X,Z), parent(Y,Z), X!=Y.
cousin(X,Y) :- grandparent(X,Z), grandparent(Y,Z), X!=Y, not sibling(X,Y).
aunt_or_uncle(X,Y) :- parent(X,Z), sibling(Y,Z).
```

Access control

Absolute basics

In your system you should have the following predicates:

- `file(X)` to state that `X` is a file.
- `principal(X)` to state that `X` is a principal (which is a fancy way of saying a *user*).
- `canRead(P, F)` to state that the principal `P` can read file `F`.

Discretionary access control

- All files have an `owner`. `owner(F,P)` states that principal `P` is the owner of file `F`.
- You can read a file if you are it's owner.
- You can read a file if `othersCanRead(F)` is true.

```
file(doc1). file(doc2). file(doc3).
principal(a). principal(b). principal(c).
owner(doc1, a). owner(doc2, b). owner(doc3, c).
%othersCanRead(doc1).

canRead(P, F) :- file(F), principal(P), owner(F, P).
canRead(P, F) :- file(F), principal(P), othersCanRead(F).
canRead(P, F) :- file(F), principal(P), owner(F, P0), saysCanRead(P0,P,F).
```

Delegated access control

- The owner of a file can state who else can read it. `saysCanRead(P1,P2,F)` states that `P1` says that `P2` can read file `F`. Remember to check that `P1` can only delegate access if they own `F`.

- A principal can delegate to other principals to decide who can read their files. `delegatesTo(P1,P2)` states that `P1` will allow `P2` to make access control decisions for them.
- Make sure that a principal whose been delegated to can also redelegate the decision to someone else!

Role-based access control

- Add roles to your access control system. For example, a principal may be happy to say that an auditor can read their files, but may not know who the auditor's currently are.
- `holds(P,R)` states that a principal `P` holds a role `R`. Update your `saysCanRead` rule to account for roles.

```
delegatesTo(b, c).
delegatesTo(c, e).
holds(a, user).
principal(d). principal(e).
holds(d, user). holds(e, common).

saysCanRead(P1, P2, F) :- saysCanRead(P1, P3, F), saysCanRead(P3, P2, F).
saysCanRead(P1, P2, F) :- owner(F, P1), holds(P1, R), holds(P2, R), P1!=P2, principal(P1), principal
(P2), file(F).
saysCanRead(P1, P2, F) :- owner(F, P1), delegatesTo(P1, P2), P1!=P2, principal(P1), principal(P2), fi
le(F).
saysCanRead(P1, P2, F) :- owner(F, P0), delegatesTo(P0, P1), delegatesTo(P1, P2), P0!=P1, P1!=P2, pri
ncipal(P0), principal(P1), principal(P2), file(F).
```

Mandatory access control

- Files have a security level.
  - `unclassified(F)` states the file is publicly available.
  - `secret(F)` states that the file is secret.
  - `topsecret(F)` states that the file is top secret.
- Security levels have an ordering
  - `unclassified` < `secret` < `topsecret`
- Principals have a clearance. `clearance(P,secret)` states that Principal `P` has a `secret` security clearance.

Implement the following security models:

Read down, write up *(Bell LaPadula)*

The *read down, write up* access control model is used to protect access to data.

- You can read a file if you have an appropriate or higher clearance than the file.
- You can write to a file if you have an appropriate or lesser clearance than the file (so that you can tell people with more clearance than you things without informing your peers).

```
unclassified(doc1).
secret(doc2).
topsecret(doc3).
file(doc4).
secret(doc4).
clearance(a,unclassified).
clearance(b,secret).
clearance(c,topsecret).
clearance(d,secret).
clearance(e,unclassified).

canRead(P,F) :- file(F), principal(P), unclassified(F).
canRead(P,F) :- file(F), principal(P), clearance(P, secret), secret(F).
canRead(P,F) :- file(F), principal(P), clearance(P, topsecret), secret(F).
canRead(P,F) :- file(F), principal(P), clearance(P, topsecret), topsecret(F).

canWrite(P,F) :- file(F), principal(P), topsecret(F).
canWrite(P,F) :- file(F), principal(P), clearance(P, secret), secret(F).
```

```
canWrite(P,F) :- file(F), principal(P), clearance(P, unclassified), secret(F).
canWrite(P,F) :- file(F), principal(P), clearance(P, unclassified), unclassified(F).
```