Overview of Software Tools

Lectures – Part 1

(Weeks 1 – 5)

COMSM0110
Joseph Hallett, Matthew Edwards, Manolis Samanis
Publicly hosted at
https://github.com/cs-uob/COMSM0085

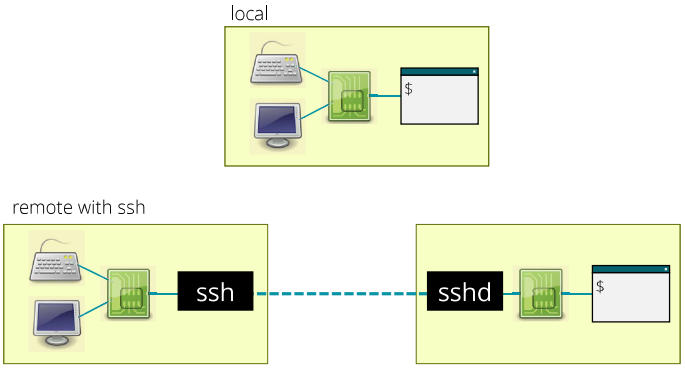
Part 1:
System administration (ssh,vagrant,apt)
Version control (git)
Shell scripting (sudo,grep,sh) and build tools (make,javac,pip)
Debugging (gdb,strace,ltrace)
Databases (mariadb, SQL)

SSH: Secure Shell

COMS10012 / COMSM0085

Software Tools

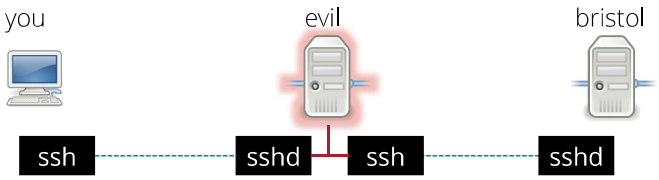
Shell



syntax

```
$ ssh [USERNAME@]HOSTNAME

...
Are you sure [y/n]?
```


man in the middle



cryptography

- SSH servers have a key pair for digital signatures.
- If you connect to a new server, SSH asks you if you're sure.
- If you connect to a server and the key has changed, you get a warning.

keys

You can create your own key too.

- convenient: you don't have to type your password anymore.
- secure: key never leaves your machine, even if you connect to an evil server.

Details in the exercises ...

servers

`seis.bris.ac.uk`
bastion host, accessible from internet, but doesn't have your files or programs

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

`it######.wks.bris.ac.uk`
lab machines, #=075637 to 075912.

Vagrant

COMS10012 / COMSM0085

Software Tools

Virtualisation

- emulate a different stack
- reproducible build environment
- cost / scalability

Software

Virtualisation:

VMware,
VirtualBox (Oracle)

bochs, qemu,
DOSbox, ...

Containers:

Docker
Kubernetes

OpenStack, rkt, ...

vagrant

Installing vagrant

Lab machines (but not seis): installed

From the web: www.vagrantup.com/download

Linux: vagrant recommends *not* using your system's package manager (but Arch seems to work).

Windows: read www.vagrantup.com/docs/installation, you may need to disable Hyper-V.

Vagrant

Host: folder with Vagrantfile (ruby)

Different providers

ssh access to guest

can share folders between host/guest

Vagrantfile

```
Vagrant.configure("2") do |config|
    config.vm.box = "generic/debian12"
end
```

box repository:
https://app.vagrantup.com/boxes/search

Start the machine

```
$ vagrant up
Bringing machine 'default' up with
'virtualbox' provider...
==> default: Importing base box
'generic/debian12'...
...
==> default: Machine booted and ready!
```


Commands

`vagrant up` start machine
`vagrant ssh` log in
`vagrant halt` stop machine
`vagrant reload` stop+start machine (for config update)
`vagrant destroy` delete machine

All commands require a Vagrantfile in the current directory.

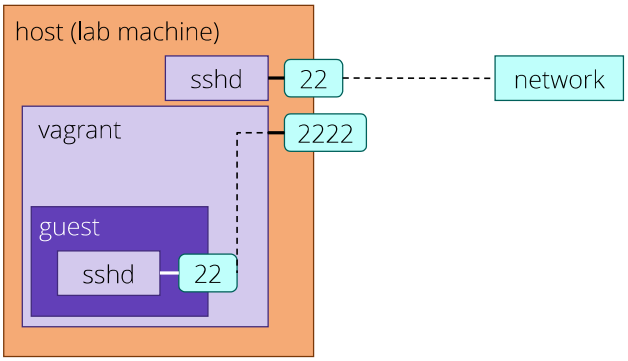
Log in

```
$ vagrant ssh
vagrant@debian12:~$
vagrant@debian12:~$ whoami
vagrant
vagrant@debian12:~$ exit
logout
Connection to 127.0.0.1 closed.
$
```


ssh

```
$ vagrant up
...
==> default: Forwarding ports...
    default: 22 (guest) => 2222 (host)
...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
    default: Vagrant insecure key detected.
Vagrant will automatically replace this with a
newly generated keypair for better security.
    default: Inserting generated public key within
guest...
```


ssh and port forwarding



keys

Remember: if you have a *secret key*, you can ssh in to a machine that has the matching *public key*.

Vagrant box (in repository) has a default public/secret key pair.

When you provision (`vagrant up`) a box, it creates a new key pair – this is more secure, and you can use it with `vagrant ssh`.

Storage

Normal use: virtual machines stored in

- Linux: `~/.vagrant.d`
- Windows: `C:\Users\NAME\.vagrant.d`

Some configuration goes in the `.vagrant` folder in the folder with the Vagrantfile.

Storage – lab machines

VMs are stored in `/tmp` and may not survive host reboots!

Also, they are not on NFS, so not visible from other lab machines.

- This is by design.
- **Treat VMs on lab machines as disposable – back up your data somewhere else!**

Debian Linux

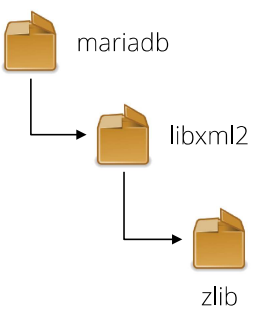
- Common, well-supported Linux distribution.
- Saves us some headaches from previous years' Alpine
- Mostly very similar – but look at slides and exercises if something in a video seems specific to Alpine.

Package managers

COMS10012 / COMSM0085

Software Tools

Packages




```
$ apt search . | wc -l
190508

$ apt search maria
mariadb-static-10.3.23-r0
mariadb-connector-c-dev-
3.0.10-r0
mariadb-bench-10.3.23-r0
mariadb-10.3.23-r0
mariadb-server-utils-10.3.23-
r0
mariadb-backup-10.3.23-r0
mariadb-doc-10.3.23-r0
mariadb-openrc-10.3.23-r0
mariadb-client-10.3.23-r0
...
```

APT

apt: Advanced Package Tool

.deb files (for Debian)

Debian/Ubuntu/Mint/...: **apt**

Alpine: **apk** (if you see us mistake **apt** for **apk** in any materials, it's because this course used to use Alpine.)

Red Hat: **rpm**

Arch: **pacman**

repositories

```
$ cat /etc/apt/sources.list
```

```
http://deb.debian.org/debian/
```

Index of /debian/dists/bookworm/main

Name	Last modified	Size
Parent Directory		-
Contents-all.gz	2023-12-10 17:34	31M
Contents-amd64.gz	2023-12-10 17:33	11M
Contents-arm64.gz	2023-12-10 17:33	11M
Contents-armel.gz	2023-12-10 17:33	9.2M
Contents-armhf.gz	2023-12-10 17:33	9.8M
Contents-i386.gz	2023-12-10 17:33	11M
Contents-mips64el.gz	2023-12-10 17:33	9.8M
Contents-mipsel.gz	2023-12-10 17:33	9.7M

Contents-<arch>.gz lists all packages for a system

finding packages

```
$ apt search [-v] [-d] STRING
```

```
$ apt info [-a] PACKAGE
```

```
$ apt list [-I] PACKAGE
```

```
$ apt [COMMAND] --help
```

update and upgrade

```
$ sudo apt update
```

Download the new list of packages, but don't install anything yet.

```
$ sudo apt upgrade
```

Upgrade all installed packages to the latest version.

installing

```
$ sudo apt install PACKAGE [PACKAGE...]
```

Installs one or more packages and their dependencies.

Vagrant:

```
$ cat /vagrant/Vagrantfile
    ...
    apt install emacs-nox
    ...
```

Find a command

By complete file path:

```
$ dpkg-query -S /bin/bash
```

The website:

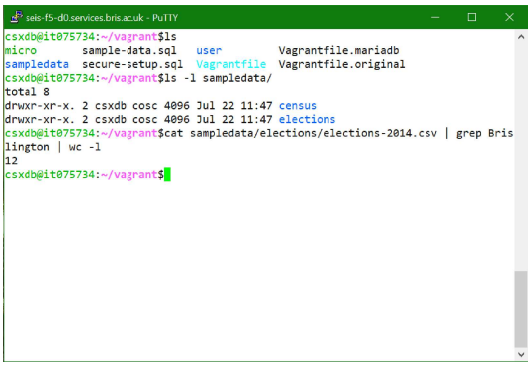
http://deb.debian.org/debian/

The shell

COMS10012 / COMSM0085

Software Tools

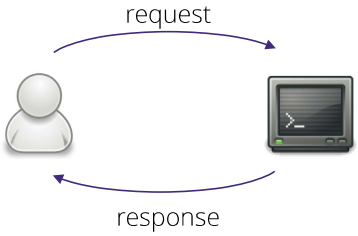
The shell



Terms

shell	xterm
terminal	rxvt
console	konsole
command line	(gnome)-terminal
(command) prompt	putty (Windows)

shell workflow



prompt

- **$** You are in a shell, most likely POSIX (sh) compatible.
- **#** You are in a root shell. With great power comes great responsibility.
- **%** You are probably in the C shell.
- **>** You are on a continuation line e.g. inside a string.

shell tricks

TAB: complete command or filename

DOUBLE TAB: show list of possible completions

UP/DOWN: scroll through history

^R text: search history for command

builtins

```
$ which ls
/bin/ls
$ which cd
$
```


options and conventions

```
$ ls
file1        file2
$ ls -l
-rwx------    1 vagrant ... 40 ... file1
-rwxr-----    1 vagrant ... 80 ... file2
$ ls -a
.          ..         file      file2
```


help

```
$ ls --help
BusyBox v1.30.1 multi-call binary.

Usage: ls [-1AaCxdLHRFplinshrSXvctu] [-w
WIDTH] [FILE]...

List directory contents

  -1      One column output
  -a      Include entries which start with
.
  ...     ...
```


manuals

```
$ man [SECTION] COMMAND
```

- On lab machines: fairly user-friendly manual.
- On alpine: programmer's manual.

Section 1 is shell commands, section 2 system calls, section 3 the C library etc.

e.g. `man 1 printf` and `man 3 printf` are different.

shell expansion

shell expansion

Separation of responsibility:

- shell deals with expanding pattern
- program deals with its arguments

shell expansion

`*` all filenames in current scope
e.g `a*` is filenames starting with a etc.

`?` single character in filename
e.g. `image???.jpg` matches `image001.jpg`

`[ab]` single character in list
e.g. `image[0-9].jpg`

`$` variable name expansion

shell quoting

`"double quotes"` turn off pattern matching keeps variable interpolation and backslashes on

`'single quotes'` turn off everything

`\*, \?, \[, \$` do not treat as pattern

example

`cp [-rfi] SRC... DEST` copy files
- **`-r`** recursive
- **`-f`** overwrite readonly
- **`-i`** ask before overwriting (interactive)

`mv [-nf] SRC... DEST` move files
- **`-n`** no overwrite
- **`-f`** force overwrite

examples

```
$ cp index.html style.css web
$ cp * web
```

in empty folder:

```
$ cp * web
cp: can't stat '*': No such file or directory
```


find files

```
$ find DIR [EXPRESSION]
```
find all files in directory (recursively) that match an expression

e.g. **`find . -name "a*"`**

Pipes 1

COMS10012 / COMSM0085

Software Tools

standard IO

Unix Philosophy

It is easier to maintain 10 small programs than one large program. Therefore,

1. Each program should do one thing well.
2. Programs should be able to cooperate to perform larger tasks.
3. The universal interface between programs should be a text stream.

source

```
#include <stdio.h>
// gives stdin etc.
// fread, fwrite, FILE* - C abstraction
#include <unistd.h>
// pulls in /usr/include/sys/unistd.h
// read, write - POSIX abstraction
#define STDIN_FILENO    0
#define STDOUT_FILENO   1
#define STDERR_FILENO   2
```


standard input/output

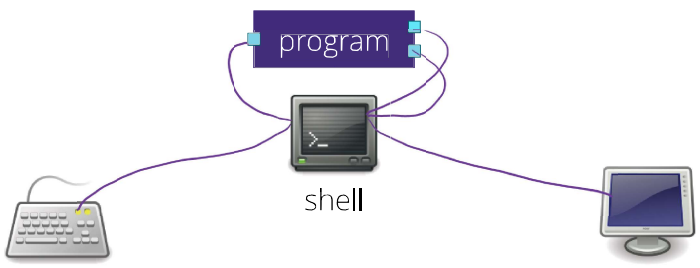
Internally, programs read(fd, buffer, size) and write(fd, buffer, size).

Each program starts with three file descriptors open:

0 = standard input
1 = standard output
2 = standard error

standard input/output

Running a program in the terminal:

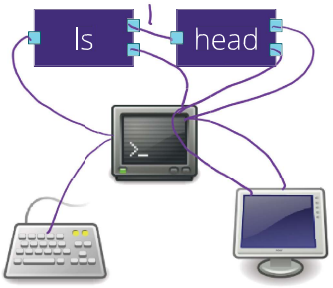


pipes

pipe

```
$ ls -1 | head
```

```
head [-n NUM]
tail [-n NUM]
```



pipe

```
$ ls -1 | grep software | sort -r
```

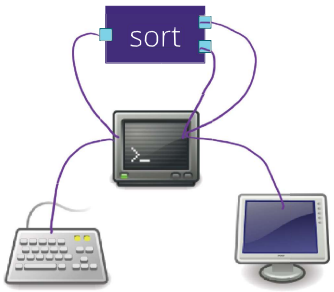
```
grep: "global regular expression parser"
sort: read all lines into buffer, sort, output
uniq: remove duplicates immediately following
      best used as: command | sort | uniq
```


grep

```
$ grep PATTERN FILENAMES

$ grep -nHi PATTERN FILENAMES

$ grep [OPTIONS] PATTERN
```


sort

```
$ sort
aaa
ccc
bbb
^D
aaa
bbb
ccc
$
```



Pipes 2

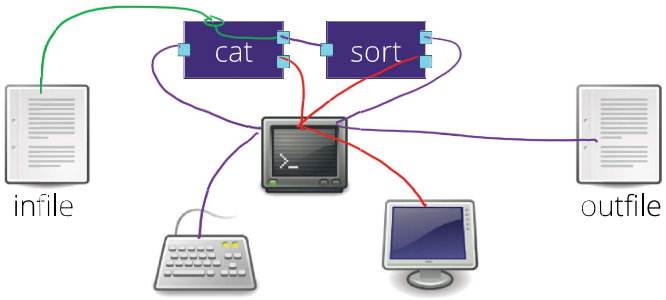
COMS10012 / COMSM0085

Software Tools

redirects

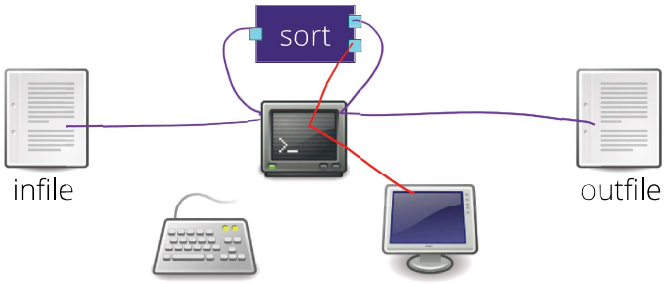
redirect

```
$ cat infile | sort > outfile
```



redirect

```
$ sort < infile > outfile
```

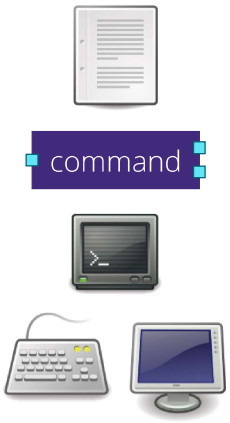


redirect

```
$ COMMAND > FILE          overwrites
FILE
$ COMMAND >> FILE         appends to
FILE
```


error redirect

```
$ COMMAND > FILE 2> FILE2
$ COMMAND > FILE 2>&1

not:
$ COMMAND 2>&1 > FILE

ignore output:
$ COMMAND > /dev/null
```



files vs streams

A program that uses a standard stream can be told to use a file instead by

- `PROGRAM < FILE` (standard input)
- `PROGRAM > FILE` (standard output)
- `PROGRAM 2> FILE` (standard error)

files vs streams

A program that expects a filename can be told to use standard input/output instead by:

- using the filename **–** (single dash), if the program supports it
- using the filename `/dev/stdin` etc., if your OS supports it

Filenames with dashes

Filenames starting with dashes are generally considered bad.

If you really want to address one (e.g. you created one by mistake), use e.g.

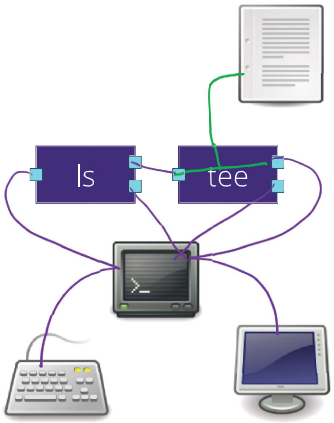
```
$ cat ./-
$ rm ./-f
```


advanced

tee

```
$ ls | tee FILE
```

`tee` takes a filename as argument and writes a *copy* of input to it, as well as to stdout



pagers

```
$ ls | less
```

`less` is a pager: it displays text on your screen, one page at a time.

- Up/Down arrows scroll,
- Space/Enter advance a page,
- / (forward slash) opens a search,
- q quits.

This takes direct control of the screen (terminal).

sed

```
$ echo "Hello World" | sed -e
's/World/Universe/'
Hello Universe
```

`sed` stands for stream editor – it can change text using a regular expression as it passes from input to output.

`s/ONE/TWO/[g]` replaces the first match for ONE (all matches, with /g) with TWO. Regular expressions are supported.

need a file, want a pipe

If PROGRAM wants a file to read from, how can I pipe something in?

```
$ PROGRAM <(SOMETHING)
```

```
$ cat <(echo "Hi")
Hi
$ echo <(echo "Hi")
/dev/fd/63
```

subshell

```
$ cat <(echo "Hi")
```

subshell to argument

```
$ COMMAND $(SOMETHING)
```

```
$ echo $(echo Hi | sed -e s/Hi/Hello/)
Hello
```

old-fashioned way, with backticks:

```
$ COMMAND `SOMETHING`
```

Git: Version control

How do we track how code changes?

Joseph Hallett

January 29, 2024

What's this all about?

- You can't write all the code you need in a day.
- Sometimes you have to share code with other people.
  - Doing that thing where you number files and stick 'FINAL.docx' is stupid.

We need a mechanism to systematically track changes.

Luckily

Software Engineers solved this problem back in the 70s.

- (and perfected it in the 2000s)

For example

Suppose *Alice* goes and writes the following program:

```
public class Hello {
   public static void main(String[] args) {
      if (args.length == 0) {
         args = new String[1];
         args[0] = "World";
      }
      for (final var name : args)
         System.out.print("Hello␣"+name+"!\n");
   }
}
```

Later she updates it...

```
public class Hello2 {
   public static void main(String[] args) {
      if (args.length == 0) {
         args = new String[1];
         args[0] = "World";
      }
      for (final var name : args)
         System.out.println("Hello␣"+name+"!");
   }
}
```

Whats changed?

A bad solution

We *could* go and track changes manually...

- Each version of the file has a different name with a number on the end
- Write a suite of tools for spotting what the differences in files are...

```
2c2
< public class Hello {
---
> public class Hello2 {
9c9
<          System.out.print("Hello␣"+name+"!\n");
---
>          System.out.println("Hello␣"+name+"!");
```

And we could store the diffs over time to keep a record of how things changed, and who changed what...

Don't work hard! Work Lazy!

Clearly managing source code like this is going to be a lot of manual work.
But we're *computer scientists*... we can automate *anything*.

So lets do that!

- Write software to do all the management of software for you
- Let it keep track of who has changed what and when
- Let the programmer step in and fix things as a last resort

Git

This is *Linus Torvalds* he made Git (and Linux).
To make a change to the kernel:

- You take a copy of the code
- Do your work
- Email a diff to Linus
- And you'll have a discussion about it over email
  - See `https://lore.kernel.org/lkml`
- If he likes it Linus will apply it to the codebase

Git is designed to be a tool to help Linus do his job

- Not designed to be user friendly
- Worse is better
- Fast for working with plaintext files (source code)
- Works well with *huge* numbers of files
- Source code isn't that complex

This is *still* how the kernel gets developed!

Git is hard

There's a manual

```
GIT(1)                    Git Manual                    GIT(1)

NAME
    git - the stupid content tracker

SYNOPSIS
    git [-v | --version] [-h | --help] [-C <path>] [-c <name>=<value>]
        [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
        [-p|--paginate|-P|--no-pager] [--no-replace-objects] [--bare]
        [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
        [--super-prefix=<path>] [--config-env=<name>=<envvar>]
        <command> [<args>]


DESCRIPTION
    Git is a fast, scalable, distributed revision control system with an
    unusually rich command set that provides both high-level operations and
    full access to internals.

    See gittutorial(7) to get started, then see giteveryday(7) for a useful
```

There are also books

`https://git-scm.com/book/en/v2` The *official* Git book.

`https://ohshitgit.com` A guide for how to get out of silly situations in Git.

You have to practice for years for it to become comfortable

- See you in the labs

Okay lets get started!

To create a *Git repo* we can use the `git init` command:

```
mkdir tutorial
cd tutorial
git init
```

```
Initialized empty Git repository in /tmp/tutorial/.git/
```

```
ls -A
```

```
.git
```

```
git status
```

```
On branch main

No commits yet

nothing to commit (create/copy files and use "git␣add" to track)
```

Lets add some code

```
cat >hello.c <<EOF
#include <stdio.h>

int main(void) {
   printf("Hello, World\n");
   return 0;
}
EOF
git add hello.c
git status
```

```
On branch main

No commits yet

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

Staging

At this point, the file `hello.c` is *staged* but it hasn't been *commited* yet.
When you *stage* a file:

- You're saying this will be part of a new commit
- You're adding the changes into Git's versioning
- But you're not saving anything
- Things can still change!

When you *commit*:

- Everything you've staged so far gets written into the history as a single change.
- With a note explaining it
- And your name associated with it
- Things *shouldn't* change
  - (techincally they still can... but it gets harder)

Lets commit!

```
git commit -m 'Initial␣commit␣of␣the␣greeting␣program.

Greets␣the␣user␣and␣then␣exits.'
```

```
[main (root-commit) 6f436f6] Initial commit of the greeting program.
 1 file changed, 6 insertions(+)
 create mode 100644 hello.c
```

```
[main (root-commit) 65a5a16] Initial commit of the greeting program.
 1 file changed, 6 insertions(+)
 create mode 100644 hello.c
```

Note

Sometimes when your on a new system you'll get a prompt to set your name and email... just follow the instructions provided. All Git commits need a name and an email address attributed to them.

```
git config --global user.name 'Joseph␣Hallett'
git config --global user.email 'joseph.hallett@bristol.ac.uk'
```

Lets make some edits

```
ed hello.c <<EOS
3c
int main(int argc, char *argv[]) {
.
4c
    for (int i=0; i<argc; i++)
        printf("Hello,␣%s\n", argv[i]);
.
wq
EOS
```

```
83
142
```

```
git add hello.c
git commit -m "Greets␣all␣the␣people␣passed."
```

```
[main 12a40a6] Greets all the people passed.
 1 file changed, 3 insertions(+), 2 deletions(-)
```

```
make hello
./hello Alice Bob
```

```
cc   hello.c -o hello
Hello, ./hello
Hello, Alice
Hello, Bob
```

One more edit...

```
ed hello.c <<EOS
4s/0/1/
wq
EOS
git add hello.c
git commit -m "Stops␣greeting␣the␣program␣itself."
```

```
142
142
[main 8147cde] Stops greeting the program itself.
 1 file changed, 1 insertion(+), 1 deletion(-)
```

```
make hello
./hello Alice Bob
```

```
cc   hello.c -o hello
Hello, Alice
Hello, Bob
```

So what have we done?

So far we've made three changes to our code: lets see what these look like in Git!

```
git log --oneline | cat
```

```
8147cde Stops greeting the program itself.
12a40a6 Greets all the people passed.
6f436f6 Initial commit of the greeting program.
```

Tags, branches and HEAD...

Commits are all *identified* by their hash...

- but you can name specific commits by using the `git tag` command
- (this is useful for marking releases or submitted versions of your code)

All commits are made to a *branch* which is a *tag*

- When the commit is made the *branch* tag is *updated* to point to the new commit at the top of *branch*.
- The *default* branch is usually called `main` (or `master`)
- (This is wrong in all important respects; but it's an okay simplification)

There is also a *special* tag called `HEAD`

- Always points to wherever your code is currently at
- Minus any unstaged work

Working with commits

Say you've made a bunch of changes to a file, but not commited them. You'd like to threw away the changes you made:

```
git checkout HEAD -- hello.c
```

Or if you've changed a lot of stuff and want to go back to clean:

```
git reset --hard HEAD # Remove all changes
git clean -dfx   # Delete all untracked files
```

```
HEAD is now at 8147cde Stops greeting the pro...
Removing hello
```

Say you'd like to go back to how the code was *before* the last commit:

```
git checkout HEAD~1
```

Say you're done looking at the code in an old state, and want to go back to working on the `main` branch:

```
git checkout main
```

Say a commit was a horrible mistake and you'd like to apply it in reverse and undo all the changes of it:

```
git revert HEAD
```

```
[main de70029] Revert "Stops␣greeting␣the␣pro...
␣Date:␣Thu␣Jan␣25␣16:33:42␣2024␣+0000
␣1␣file␣changed,␣1␣insertion(+),␣1␣deletion(-)
```

So far we've just been working on our own code

We've just been tracking local changes...

- But Git was made to let people share code

Lets make it more complex!

Rather than making *our own* new repo:

- Let's take a copy *or clone* someone else's
- And let's share those changes with them

What do we mean by *decentralized*?

We said Git was a *decentralized version control system*

- As opposed to a *centralized* one like SVN/CVS

What this means in practice is that your local repo *should* have the complete history of the repo.

- And should be able to function as a master copy of the repo.
- (Again this is a simplification but go with it...)

Let's pretend Alice has a Git repo of their course groupwork in `~alice/coursework/`

- Bob wants to collaborate with Alice and get their own copy

So Bob runs...

```
git clone ~alice/coursework ~bob/coursework
```

```
Cloning into '~bob/coursework'...
done.
```

```
cd ~bob/coursework
git log --oneline
```

```
af3818c States true facts about this course
7eb311c First draft of the coursework
```

```
make coursework
```

```
cc -O2 -pipe -o coursework coursework.c
```

```
./coursework
```

```
Softwaer tools is cool!
Hello World!
```

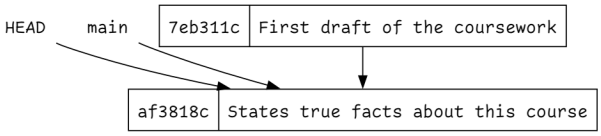
Oh no: a mistake!

Bob can fix that for Alice!

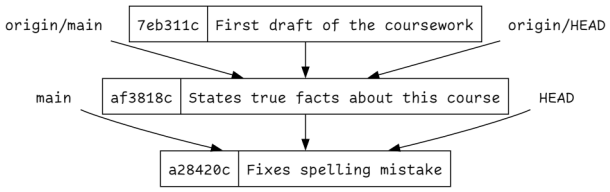
```
$ ed coursework.c
124
4c
   printf("Software␣tools␣is␣cool!\n");
.
wq
124

$ git add coursework.c

$ git commit -m "Fixes␣spelling␣mistake"
[main a28420c] Fixes spelling mistake
 1 file changed, 1 insertion(+), 1 deletion(-)
```

So now

On Alice's repo



On Bob's repo



So how do we get Bob's changes back to Alice?

There are multiple ways!

Bob can send Alice their changes patch based

Alice can pull Bob's changes pull based

Patch based approach (Bob's end)

This is how the *Linux Kernel* and many other open source projects manage commits.
Bob starts by preparing a *patch*

```
$ git format-patch origin/main \
    --to=alice@bristol.ac.uk
0001-Fixes-spelling-mistake.patch
```

And sends it to Alice

- (check out `=git send-email=`!)
  - (You'll probably need to configure `sendmail` for that...)

```
From a28420cd5c45d06c9a51625d5a03c37bb77e2ca9 Mon Se
From: bob <bob@bristol.ac.uk>
Date: Tue, 22 Nov 2022 11:53:04 +0000
Subject: [PATCH] Fixes spelling mistake
To: alice@bristol.ac.uk

---
 coursework.c | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

diff --git a/coursework.c b/coursework.c
index 2e191f8..8927b2f 100644
--- a/coursework.c
+++ b/coursework.c
@@ -1,7 +1,7 @@
 #include <stdio.h>

 int main(void) {
-    printf("Softwaer tools is cool!\n");
+    printf("Software tools is cool!\n");
     printf("Hello World!\n");
     return 0;
 }
--
2.38.1
```

Patch based approach (Alice's end)

Alice reviews Bob's patch and, if they like it... applies it to their tree.

```
$ git am ../bob/0001-Fixes-spelling-mistake.patch
Applying: Fixes spelling mistake

$ git log --oneline
575dcde Fixes spelling mistake
af3818c States true facts about this course
7eb311c First draft of the coursework
```

The ID is different however to Bob's tree for the latest commit

- (because *Alice* commited it).

If Bob wants to keep IDs in sync with Alice they need to re-`clone`

- (or `git fetch origin`).

Aside

Technically `git am` is actually a couple of git commands rolled into one...

- First it runs `git apply` with the patch to stage all the changes it will make
- Then it runs `git commit` with the commit message also supplied in the patch

There are a *lot* of git commands that are really lower level commands chained together—watch out for them!

The alternative

Some people **hate** the patch workflow.

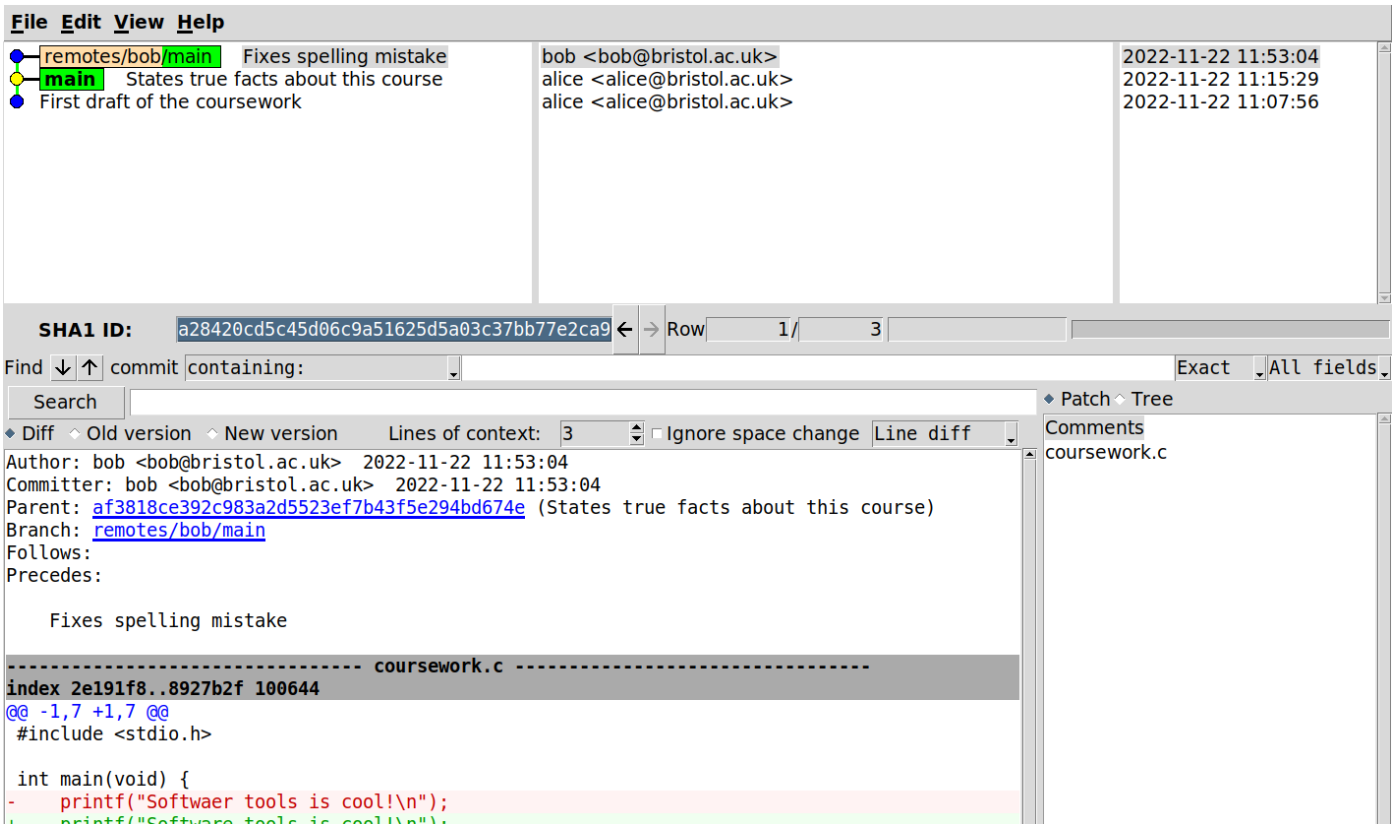
- Who configures `sendmail` nowadays?

The alternative is to let Git do the work for you and trust the other person.
Bob tells Alice they fixed a mistake.

- Alice adds Bob's repo as a `remote`

```
$ git remote add bob ~bob/coursework

$ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 255 bytes | 127.00 KiB/s, done.
From ~bob/coursework
 * [new branch] main     -> bob/main
```

Alice inspects Bob's changes...



And if they're happy...

```
$ git pull bob main
From ~bob/coursework
 * branch       main     -> FETCH_HEAD
Updating af3818c..a28420c
Fast-forward
 coursework.c | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

$ git log | cat
commit a28420cd5c45d06c9a51625d5a03c37bb77e2ca9
Author: bob <bob@bristol.ac.uk>
Date: Tue Nov 22 11:53:04 2022 +0000

   Fixes spelling mistake

commit af3818ce392c983a2d5523ef7b43f5e294bd674e
Author: alice <alice@bristol.ac.uk>
Date: Tue Nov 22 11:15:29 2022 +0000

   States true facts about this course
```

...

Aside

Again the `git pull` command is really a composite. The following is equivalent:

```
$ git fetch bob

$ git merge --ff bob/main
Updating af3818c..a28420c
Fast-forward
 coursework.c | 2 +-
 1 file changed, 1 insertion(+), 1
```

Locallity matters

So far we're dealing with files on a local filesystem.

- We cloned from a direct path to another user's Git repo
- Git *in-the-plumbing* is just a protocol for how you move files about and what you stick in that `.git` folder and how you manipulate them.

We've been accessing those files via a local shell:

- But Git doesn't care
- HTTP has a filesystem API
- SSH lets you access remote shells

Wouldn't it be *neat* if instead of having to do everything locally we could have a *centralised forge* where we go and get all our changes and send them back when we're done?

Github

Github is **not** git.

Github gives you a *centralised* remote (called a *forge*, and usually some build automation and issue tracking software):

- You can sign up for an account
- Set up access for users
- And then centrally send commits to everyone with the `git push` command
- Can host a project page and build infrastructure too

Github is owned by Microsoft:

- Some people don't like that
- Some ~~questionable~~ *naughty* behaviour surrounding AI and Open Source

Alternatives:

`https://bitbucket.org` Owned by Atlassian

`https://gitlab.com` Can self-host

`https://sr.ht` Owned by *Drew DeVault*... costs money (but it's really good)

Self host? All you need is a server (search for *bare repositories* to find out how)

To use a forge

In Bob's repo

```
$ git remote set-url origin \
  git@github.com:alice/coursework

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

nothing to commit, working tree clean

$ git push
Everything up-to-date
```

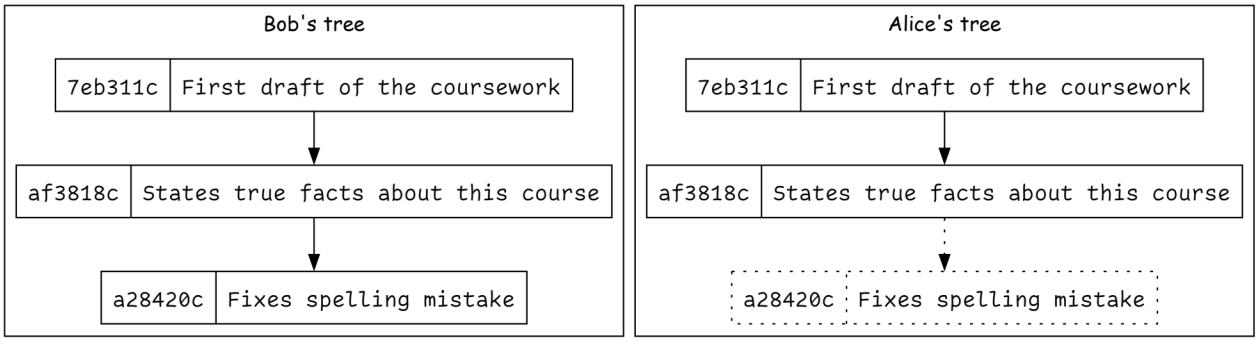
In Alice's repo

```
$ git remote -v
remote  git@github.com:alice/coursework (fetch)
remote  git@github.com:alice/coursework (push)

$ git pull remote main
From git@github.com:alice/coursework
 * branch       main     -> FETCH_HEAD
Updating af3818c..a28420c
Fast-forward
 coursework.c | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

So what happens when things go wrong?

When *Alice* pulled *Bob*'s changes earlier they could be *fast-forwarded*.

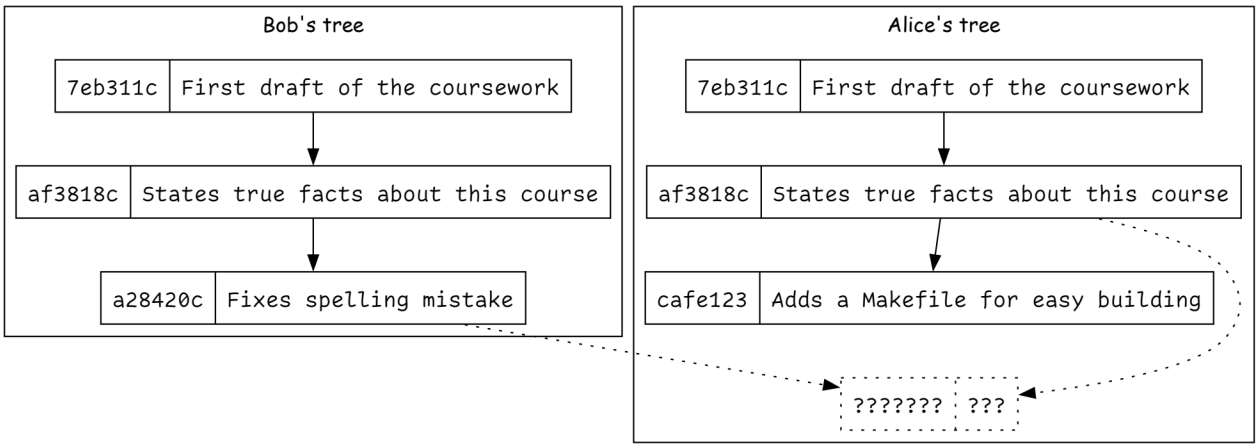
- This means that the changes could be pulled straight across and copied into Alice's tree.



Busy, busy, busy...

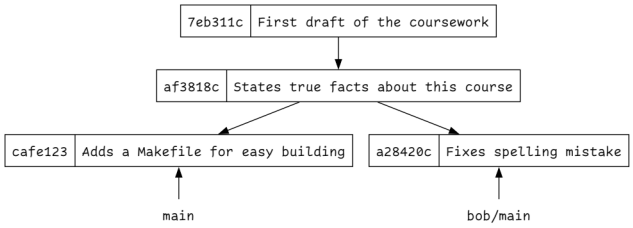
What about if *Alice* has been busy and made some commits of their own.

- The fast forward can't happen now becuase the trees have *diverged*



Merging

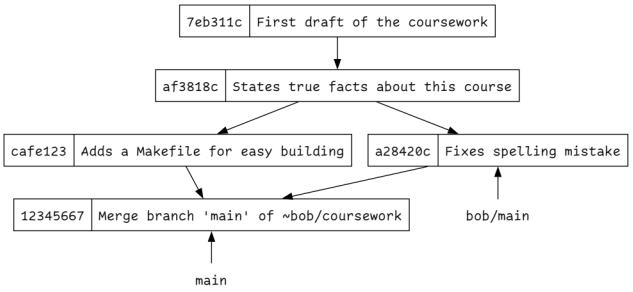
From Alice's point of view this is what the trees look like




```
git merge --no-ff bob/main
```

```
hint: Waiting for your editor to close the file.
Merge made by the 'ort' strategy.
 Makefile | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 Makefile
```

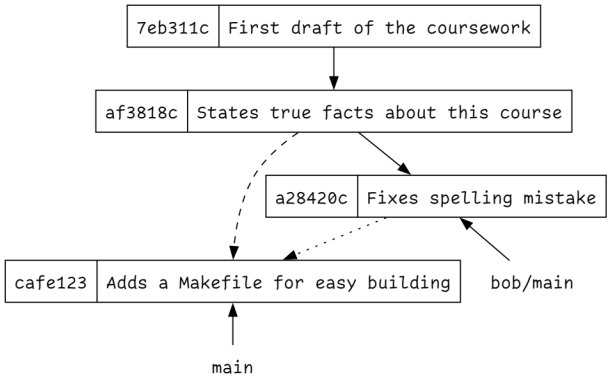
The simplest approach is to do a *merge* and add a commit explicitly merging the changes from both paths of the tree



(But normally it'll be smart and spot that you changed different files and still do the *fast-forward*...)

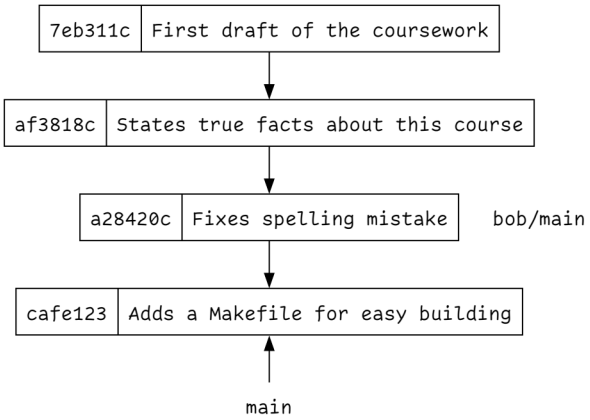
Rebasing

Alternatively Alice could do a bit of time-travelling... Lets *pretend* that Alice's commit came after Bob's:




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

Then we can *fast-forward* as before through *Bob's* change and then replay *Alice's* new commit after.



Now we get a nice neat straight line tree again!

Merging vs Rebasing

Merging is simpler *conceptually*...

- ...but messy

Rebasing is neater

- ...but complicated and prone to failure
- There are some other neat tricks you can do that make it much better (e.g. `git bisect`)

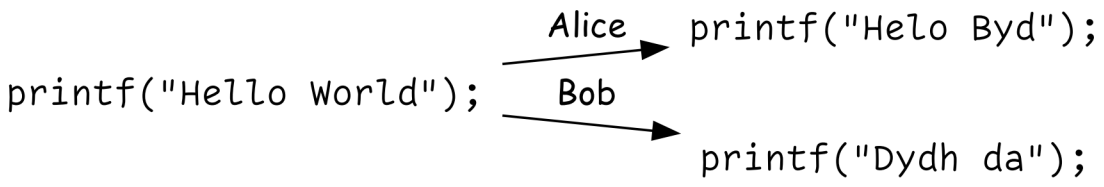
You (and your employers) will have an opinion

- Do that.
- It really doesn't matter
- (I slightly prefer the merge version but I switch back and forth...)

So far easy!

So far our merges have been *easy*.

- Alice and Bob have made edits to different files
- Changes have all been able to be done by Git automatically.
  What happens if Alice and Bob *both change the same lines in the same file*?




```
$ git merge bob/main
Auto-merging coursework.c
CONFLICT (content): Merge conflict in coursework.c
Automatic merge failed; fix conflicts and then commit the result.
```

Lets fix the conflict

Git has discovered there are two sets of changes and it can't work out which is the one to go with...
If we *follow Git's instructions* `coursework.c` looks like:

```
#include <stdio.h>

int main(void) {
<<<<<<< HEAD
  printf("Helo Byd\n");
=======
  printf("Dydh da\n");
>>>>>>> bob/main
  return 0;
```

Fix up the file and then run `git add` / `git commit` when it looks good...

- Don't just delete one side of it.
- ...Seriously... I've seen people fired for that.

```
$ git add coursework
$ git commit
[main 16d3aa6] Merge remote-tracking branch 'bob/main'
```

Wrap up of remotes

- Use `git remote` and `git clone` to work with other people
- Use `git fetch` or `git pull` or patch files to get other peoples work
- Use `git merge` or `git rebase` to integrate changes
- Use `git push` to send work back to a forge
- Merge conflicts are a pain but you have to deal with them

Merge tools

If you find yourself dealing with merge conflicts regularly... there are tools that help you work with them

`https://meldmerge.org` Good tool for dealing with merges

(I use Emacs.)

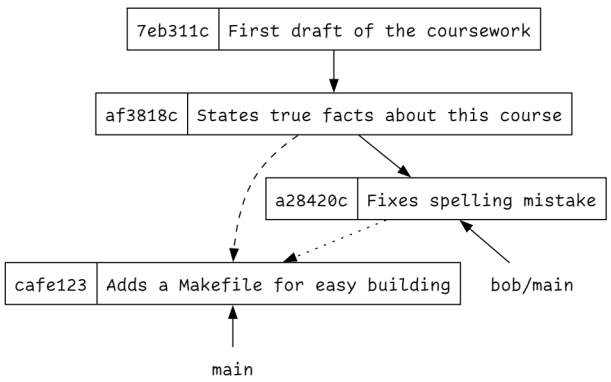
Forking timelines

So far we've had a local copy ofthe code and a remote one...

- But why should Bob have all the fun?
- What if Alice wants to have her own versions of the code with spearate sets of changes?

Lets talk about *branches* and show some tricks for working with them!

Last time...



We had this situation where *Alice* and *Bob*'s trees had diverged...

- ...but they had a shared history
- ...and we could bring them back together

But why do we need to restrict this to just other people's trees?

Branch refresher

A branch is a tag to the last commit in a trail of commits that gets updated every time you make a new commit to that branch.

Branching

The plan

Lets aim to keep the `main` branch clean

- The main branch always works

When we do some work we take a branch off of `main`

- (or possibly some other sensible place)
- Do the work...
- Merge back in when done.

Lets give it a go!

```
git branch new-feature main
git checkout new-feature
```

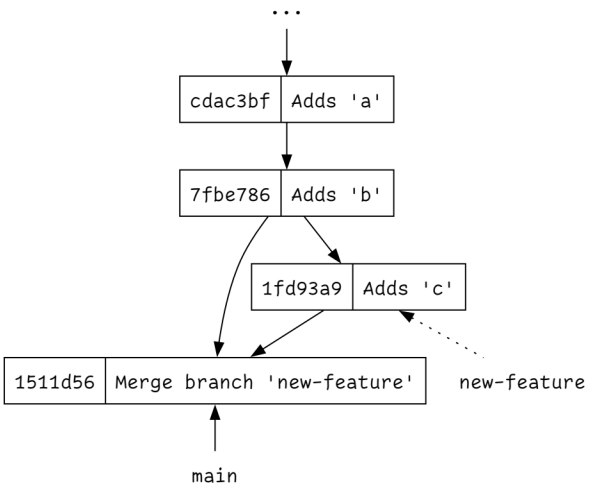
```
Switched to branch 'new-feature'
```

```
touch c; git add c; git commit -m 'Adds␣c'
```

```
[new-feature 1fd93a9] Adds c
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 c
```

```
git checkout main; git merge new-feature
```

```
Switched to branch 'main'
Merge made by the 'ort' strategy.
 c | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 c
```

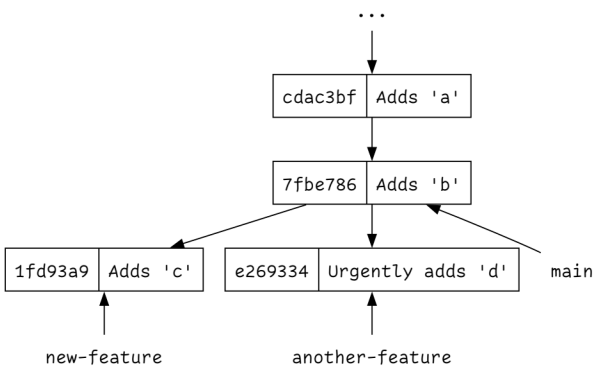


Why on earth would you do this?

It starts to come in handy when you're working on *multiple features* at once.
Say whilst you're developing your `new-feature`, your friend needs you to urgently work on `another-feature`...

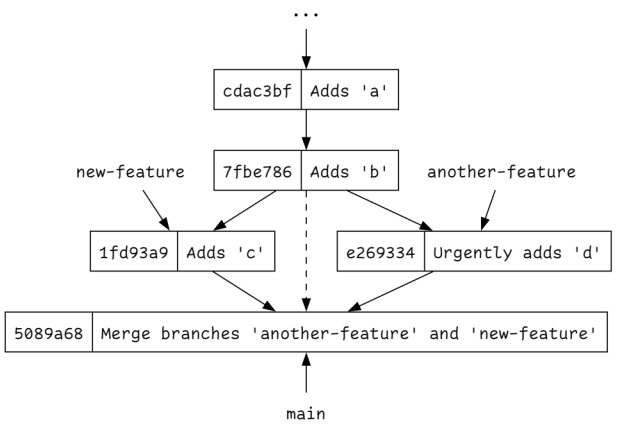
- You don't want to merge `new-feature` in yet though because you're still working on it.
- You dont want to add unrelated code for a `new-feature` in with the work for `another=feature`.

```
$ git branch another-feature 7fbe786

$ git checkout another-feature
Switched to branch 'another-feature'

$ touch d
$ git add d; git commit -m 'Urgently␣add␣d'
[another-feature e269334] Urgently add d
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644
```



Merge them all!

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

$ git merge --no-ff another-feature new-feature main
Merge made by the 'octopus' strategy.
 c | 0
 d | 0
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 c
 create mode 100644 d
```



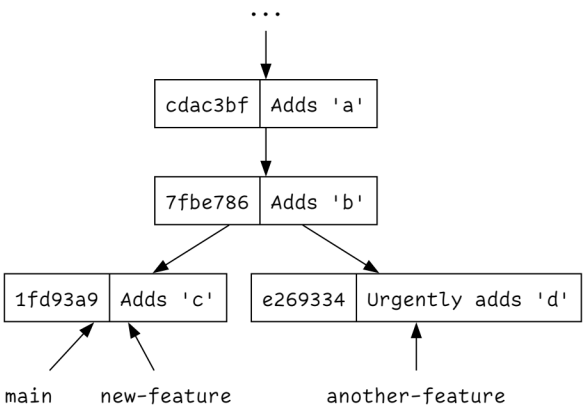
Normally you wouldn't bother with the `--no-ff` and the dashed line would disappear!

- Sometime you like the extra info (especially when teaching)
- But normally it's just noise...

Well except...

Normally you *wouldn't* do this

- What if merging `another-feature` breaks something in `new-feature`?
- It would be nice to test things before merging!
- But `new-feature` doesn't have the work now merged into `main`!



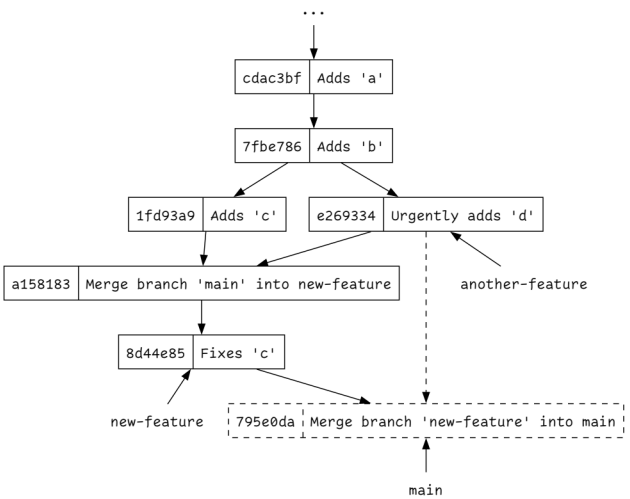
We could merge `main` into `new-feature`

We could try merging main into `new-feature`...

- Test and see if any extra changes are needed...
- Add extra commits as required
- Then merge the `new-feature` *back into* `main`

Still a bit messy as we're going to get *at least one* merge

- (assuming we don't disable fast-forwarding)



Instead we could rebase

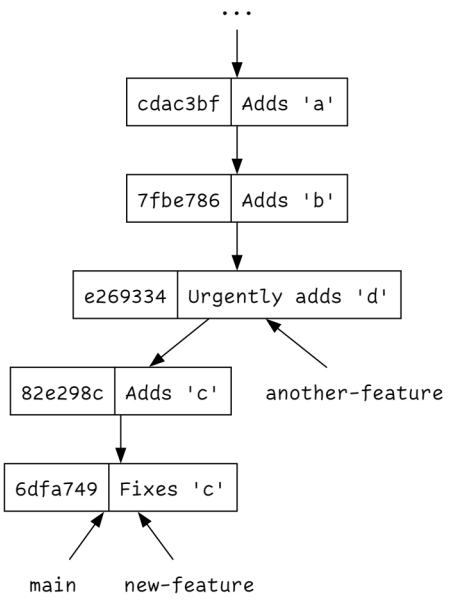
What *I'd* normally do is *rebase* new feature on `main`

- Essentially rewrite history so it looks like `new-feature` was done after the merge of `another-feature`
- Fix any conflicts then and there as part of the original `Adds 'C'` commit
  - (potentially changing its ID)
- *Then* re-test and fix issues and commit
- Then merge back into `main`.

```
$ git rebase main new-feature
Successfully rebased and updated refs/heads/new-feature.

$ git checkout new-feature
Already on 'new-feature'
```

(I always get the rebase command the wrong way round)

- (Seriously, it took 3 attempts...)
- (Always make a backup before rebasing)



We can do more with rebase!

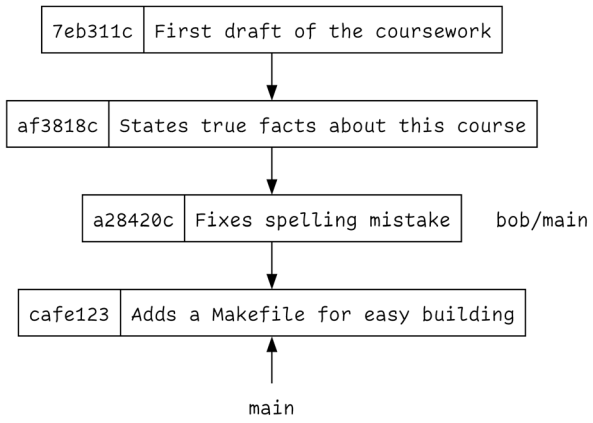
Suppose we were going to send `new-feature` as a series of patches to merge by a project maintainer

- `git format-patch` would generate one patch for each commit
- And thats fiddly for the maintainer to apply
- And it'd mean that we have commits where the whole thing is broken before we fixed C.

Rebase lets us edit the repository history!

So once we've done the fix we can rebase a branch interactively and decide what to do

```
$ git rebase -i main new-feature
[detached HEAD 7d2180a] Adds c, and fixes it...
 Date: Fri Nov 25 14:31:08 2022 +0000
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 c
[detached HEAD 5af45b8] Adds c, and fixes it...
 Date: Fri Nov 25 14:31:08 2022 +0000
 1 file changed, 1 insertion(+)
 create mode 100644 c
Successfully rebased and updated
refs/heads/new-feature.
```



This is considered a professional courtesy amongst software engineers

- Also good for hiding all those *argh I broke it* commits
- And removing swearing before you send it to the customer

Warning!

Being too clever with rebase will break your repo

Sometimes in unfixable ways

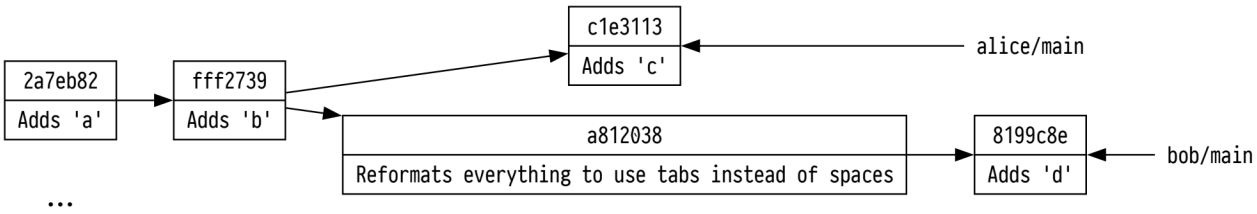
- *Always* backup before being clever
- `rebase` is considered *advanced* Git

One last trick...

Suppose Bob has done some interesting work on their `main` branch, and also some less interesting work,.

- They've fixed some bugs,
- But they've also switched all your files from using spaces to tabs

How do you `cherry-pick` the things you want and ignore the things you don't?



`git cherry-pick`

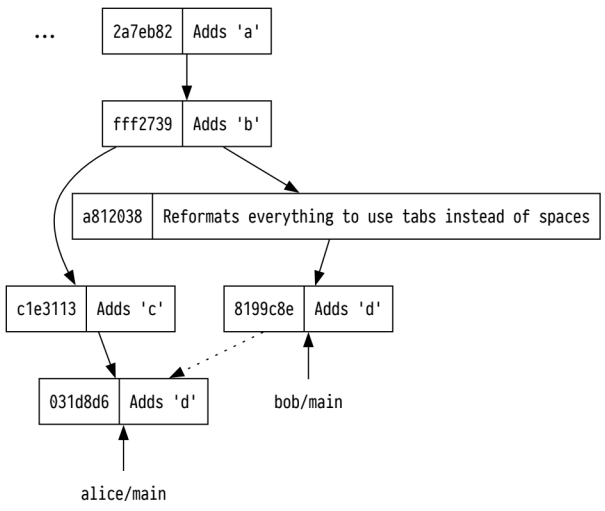
```
$ git cherry-pick 8199c8e
[main 031d8d6] Adds 'd'
 Date: Mon Nov 28 09:10:43 2022 +0000
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 d
```

Why would you do this?

- Git will generally be *somewhat* smart about how it copies the work over
- If it needs more commits it'll pull them too
- If you merge later, Git will be *somewhat* smart about where things came from and *maybe* not cause a conflict

Internally this is just a rebase...

- But Git hides a lot of the complexity
- ...actually everything in Git is really just a *rebase* with a nicer UI `;-)`



Wrap up

Right that's Git!

- There are infinitely more things you can do with it
- ...but *hopefully* this is 90% of what you'll *normally* do

Golden rules

- Do not break the build
- Write helpful log messages
- Rebase with fear (but you do have to do it sometimes)

DO NOT BREAK THE BUILD

- If you did it in the more professional places I worked; you stayed late til it was fixed
- If you did it in a startup I worked; you stayed late til it was fixed and you owed everyone a beer and got called names
- If you do it in IBM or Google; you were fired.
  - At least according to my old Prof (Awais Rashid)

Permissions

Joseph Hallett

January 12, 2023

University of BRISTOL

In the beginning there was `root`...

And it was good.

The `root` user is all powerful.

- The *super user*
- The *system administor*
- UID 0

And so root beget `init` which beget `login`...

And so other users came to pass

- Each less powerful than the original root

For inside the password file...

In the bowels of the computer's configuration directory `/etc/`:

```
$ grep -Ev '^_' /etc/passwd | column -ts :
```

Username	Password	UID	GID	GECOS	Home Directory	Shell
root	*	0	0	Charlie &	/root	/bin/ksh
daemon	*	1	1	The devil himself	/root	/sbin/nologin
operator	*	2	5	System &	/operator	/sbin/nologin
bin	*	3	7	Binaries Commands and Source	/	/sbin/nologin
build	*	21	21	base and xenocara build	/var/empty	/bin/ksh
sshd	*	27	27	sshd privsep	/var/empty	/sbin/nologin
www	*	67	67	HTTP Server	/var/www	/sbin/nologin
nobody	*	32767	32767	Unprivileged user	/nonexistent	/sbin/nologin
joseph	*	1000	1000	Joseph Hallett,,,	/home/joseph	/usr/local/bin/bash

See `man 5 passwd` or your OS's manual pages.

(Can anyone spot what OS I use?)

And inside the group file....

```
$ grep -Ev '^_' /etc/group | column -ts :
```

Groupname	Password	GID	Members
wheel	*	0	root,joseph
daemon	*	1	daemon
kmem	*	2	root
sys	*	3	root
tty	*	4	root
operator	*	5	root
bin	*	7	
wsrc	*	9	joseph
users	*	10	
auth	*	11	
games	*	13	
staff	*	20	root,joseph
wobj	*	21	joseph
sshd	*	27	
guest	*	31	root
utmp	*	45	
crontab	*	66	
www	*	67	
network	*	69	
authpf	*	72	
dialer	*	117	
nogroup	*	32766	
nobody	*	32767	
joseph	*	1000	

Something very similar

- Each group can have *multiple* members
- No passwords ever actually listed
  - (They're in `/etc/shadow`)

For all files were owned by a user and a group...

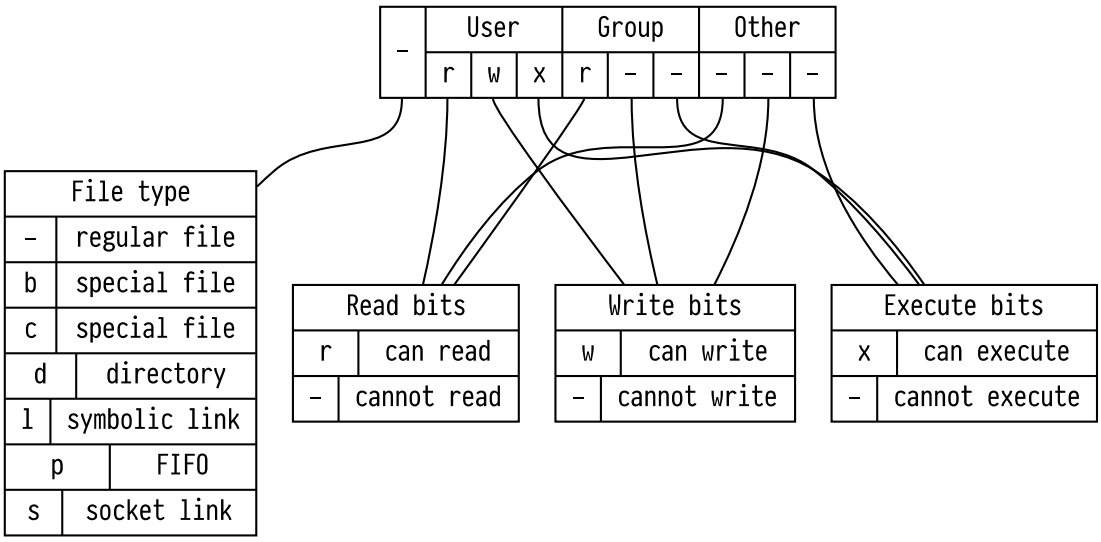
`ls -loh /etc/`

Permissions		UID	GID	File flags	Size	Filename	
drwxr-xr-x	5	root	wheel	-	512B	May 20 2022	ConsoleKit
drwxr-xr-x	2	root	wheel	-	512B	Nov 25 13:25	ImageMagick
drwxr-xr-x	7	root	wheel	-	512B	Nov 16 20:19	X11
-rw-r--r--	1	root	wheel	-	20.5K	Nov 6 12:41	abcde.conf
drwx------	2	root	wheel	-	512B	Nov 16 19:39	acme
-rw-r--r--	1	root	wheel	-	1.7K	Sep 22 19:03	adduser.conf
drwxr-xr-x	2	root	wheel	-	512B	Nov 16 19:39	amd
-rw-r--r--	1	root	wheel	-	271B	Oct 30 19:14	anthy-conf
drwxr-xr-x	3	root	wheel	-	512B	Nov 25 13:27	apache2
-rw-r--r--	1	root	wheel	-	1.8K	Nov 14 10:34	authentication$_{\texttt{milter.json}}$

…

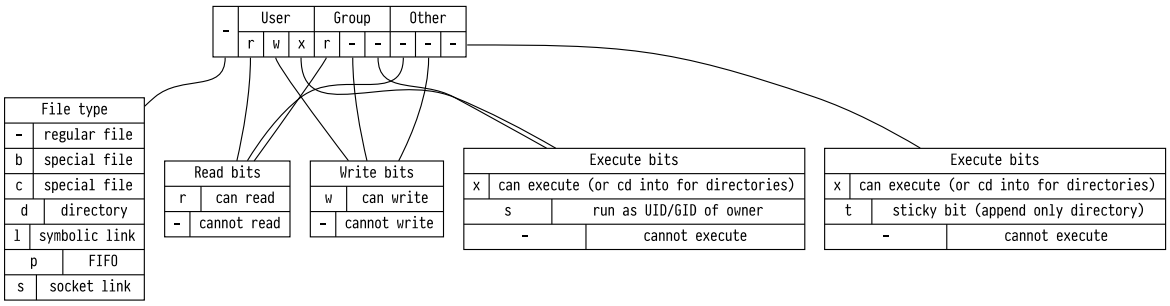
UNIX Discretionary Access Controls

And the owner of each file could set the *permissions* for each file



Actually its a *bit* more complex

And the owner of each file could set the *permissions* for each file



And, honestly, on some systems/filesystems it gets *even more* complex

- But this is 99.99% of everything you'll ever see or use

So what are those weird extra bits for

The *sticky* bit `t` is mostly for log directories and temporary directories

- You should be able to append to log files, but not delete them

The `setuid/setgid` bits are used for privilege separation.

For example how do you update your password?

Passwords are normally stored *securely* bin the shadow file `/etc/shadow`, or equivalent

- But I use OpenBSD...

```
ls -l /etc/spwd.db
```

-rw-r—- 1 root _shadow 40960 Dec 22 15:03 /etc/spwd.db

Changing passwords

The `passwd` program changes your password:

```
ls -l $(command -v passwd)
```

```
-r-sr-xr-x 1 root bin 21208 Jan 12 03:08 /usr/bin/passwd
```

Other useful `setuid` programs

- `su` switch to user (by default root) with *their* password
- `sudo` switch to user *if the sysadmin says you're allowed to* with *your* password
- `doas` modern rewrite of `sudo` with less bugs and Spiderman references

See `man su` or `man sudo` or `man doas`...

- Or *Michael W. Lucas*'s excellent *Sudo Mastery*
- (You can do a lot with `sudo...`)

Generally `setuid` programs are dangerous and you want to use them extremely carefully!

Sysadmining

How do you change who owns a file?

```
ls -l exam
```

```
-rw-r--r-- 1 joseph joseph 0 Jan 12 11:49 exam
```

```
chown joseph:staff exam
# Alternatively...
chown :staff exam
ls -l exam
```

```
-rw-r--r-- 1 joseph staff 0 Jan 12 11:49 exam
```

(See `man 1 chown`)

How do you change a file's permissions

```
chmod go-wx exam
ls -l exam
```

```
-rw-r--r-- 1 joseph staff 0 Jan 12 11:49 exam
```

Footnote

Some people like to use octal (base 8) to express permissions, where r=4, w=2, x=1...
Instead of saying `go-wx` to remove `w` and `x` bits from the group and other permissions they'll say:
`chmod 744 exam`
I suggest you give these people a wide berth.

- (but you should know how to do it)

Recap

Systems have users!

- The UNIX DAC lets you set file permissions!
- `setuid` and `setgid` programs exist!
- Root's firstname is *Charlie*!

`chmod` to change permissions

`chown` to change file owners

One more thing...

Traditionally the `root` user can do anything...
In *most* modern operating systems this has been split up a bit more

- For example Linux uses *capabilities* to set what things any user can do
- ...and *namespaces* to allow multiple root users with different capabilities

`man 7 capabilities` if you want to know more

- ...but *most of the time* you won't need to know about them...
- Unless you use Docker...

This is a lie, you really *should* know about them... but unless you're routinely in the habit of writing sysadmin tools or privileged programs you *won't normally* need to touch them. Hey, I'm a security researcher I think this stuff is fascinating but other people don't. Don't get it meself: it isn't *that* complex but hey ho. I tried.

Shellscripting: Absolute Basics

Joseph Hallett

January 12, 2023

Whats all this about?

I've written a *lot* of code over the years:

Assembly, C and Java as an engineer
Commonlisp for my own projects
Haskell to build compilers
PostScript to draw really efficient diagrams
LaTeX to publish books
...several a dozen other things too

Which language have I written the most code in?

Which language do I use to solve most tasks?

Which language do I like the least?

Shellscripting!

Normally we type commands for the terminal on a commandline...

- But we can automate them and stick them into scripts

Anything you have to do more than once...

Write a script for it!

- Saves a tonne of time
- Often easier than writing a full program

For example...

```
#! /bin/sh
GREP=grep
if [ $(uname) = "OpenBSD" ]; then
    # Use GNU Grep on OpenBSD
    GREP=ggrep
fi

${GREP} -Pi "^${1}$" /usr/share/dict/words
```

Sometimes I cheat at *Wordle*:

- I want to know a word that matches a regex exactly
- I can search the system dictionary file at `/usr/share/dist/words`
- `grep` can do the search, but I need to explicitly specify GNU Grep on systems where it isn't the default

```
knotwords 'st[^aeo]pid'
- stupid
```

Or for example...

```
#! /usr/bin/env bash
if [ $1 = "should" -a $2 = "also" -a $3 = "run" ]; then
  shift 3
  gum confirm "Run 'doas $*'?" && doas $*
elif [ $1 = "should" -a $2 = "also" -a $3 = "remove" ]; then
  gum confirm "Delete '$4'?" && doas rm -fr "${4}"
else
  2>&1 printf "WARNING You should read the commands you"
  2>&1 printf "paste more carefully\n"
fi
```

Sometimes when I upgrade my computer it tells me to delete some files or run some commands:
`You should also run rcctl restart pf`
Copying and pasting the precise text is a pain...

- Can I just copy the whole line and run that?

(Of course I can... should I though?)

Or for a further example...

```
#! /usr/bin/env bash
# Fix kitty
/usr/local/opt/bin/fix-kitty

# Update sources
cd /usr/src && cvs -q up -Pd -A
cd /usr/ports && cvs -q up -Pd -A
cd /usr/xenocara && cvs -q up -Pd -A
```

After I upgrade my computer I need to run a couple of standard commands.

- I can never remember them
- Batch them up!

So whats this really about?

Shellscripting is about automating all those tedious little jobs

- Byzantine syntax (based on shell commands)
- Awful for debugging
- Requires magical knowledge
- Probably the most useful thing you'll ever learn

Luckilly we have help

Shell scripting is somewhat magical, and there are *lots* of gotchas...

https://www.shellcheck.net

Wonderful tool to spot unportable/dangerous things in shell scripts

- Commandline tool available
- Run it on *everything* you ever write
- `shellcheck` is great

```
shellcheck `command -v knotwords`
In /home/joseph/.local/bin/knotwords line 2:
GREP=grep
^--^ SC2209 (warning): Use var=$(command) to assign output (or quote to assign string).


In /home/joseph/.local/bin/knotwords line 3:
if [ $(uname) = "OpenBSD" ]; then
     ^------^ SC2046 (warning): Quote this to prevent word splitting.

For more information:
  https://www.shellcheck.net/wiki/SC2046 -- Quote this to prevent word splitt...
  https://www.shellcheck.net/wiki/SC2209 -- Use var=$(command) to assign outp...
```

So how do you write one?

Start the file with the *shebang* `#!` then the path to the interpreter of the script plus any arguments:

For portable POSIX shellscripts `#! /bin/sh/`

For less portable BASH scripts `#! /usr/bin/env bash`

Then

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

The rest of the file will be run by the interpretter you specified

- or `sh my-script.sh` if you don't want to/can't mark it executable.

(Hey this is also why Python scripts start `#! /usr/bin/env python3`)

Why `env`?

Hang on, you might be saying, I know that bash is always in `/bin/bash`... can I just put that as my interpretter path?

Yes, but...

In the beginning `/bin` was reserved for just *system* programs

- and `/usr/bin` for admin installed programs
- and `/usr/local/bin` for locally installed programs
- and `/opt/bin` for optional installed programs
- and `/opt/local/bin` for optional locally installed programs
- and `~/.local/bin` for a *users* programs
- ...oh and sometimes they're even mounted on different disks!

This is *kinda* madness.

- So *must* Linux systems said look we'll just stick everything in `/bin` and stop using multiple partitions
- But some said no it should be `/usr/bin`, one said `/Applications/`, and others stuck them in `/usr/bin` but symlinked them to `/bin`
- And on some systems users grew fed up of the outdated system `bash` and compiled their own and installed it in `~/.local/bin` ...
- ...and ever tried using Python `venv`?

env

```
ENV(1)                    General Commands Manual                   ENV(1)

NAME
     env – set and print environment

SYNOPSIS
     env [-i] [name=value ...] [utility [argument ...]]

DESCRIPTION
     env executes utility after modifying the environment as specified on the
     command line.  The option name=value specifies an environment variable,
     name, with a value of value.
```

What `env` does is look through the `PATH` and tries to find the program specified and runs it.

...Path?

There is an environment variable called `PATH` that tells the system where all the programs are:

- Colon separated list of paths

If you want to alter it you can add a line like to your shell's config

```
export PATH="${PATH}:/extra/directory/to/search"
```

Your shells config is *possibly* in `~/.profile` but it often varies... check the `man` page for your `${SHELL}`
Also some shells have different syntax (e.g. `fish`)...

```
$ tr ':' $'\n' <<< $PATH
/home/joseph/.local/share/python/bin
/bin
/usr/bin
/sbin
/usr/sbin
/usr/X11R6/bin
/usr/local/bin
/usr/local/sbin
/home/joseph/.local/bin
/usr/local/opt/bin
/usr/games
/usr/local/games
/usr/local/jdk-17/bin
/home/joseph/.local/share/go/bin
```

Basic Syntax

Shell scripts are written by chaining commands together

- `A; B` run A then run B
- `A | B` run A and feed its output as the input to B
- `A && B` run A and if successful run B
- `A || B` run A and if not successful run B

How does it know if its successful?

Programs *return* a 1 byte exit value (e.g. C main ends with `return 0;`)

- This gets stored into the variable `${?}` after every command runs.
- `0` indicates success (usually)
- `>0` indicates failure (usually)

This can then be used with commands like `test`:

```
do_long_running_command
test $? -eq 0 && printf "Command succeeded\n"
```

Or the slightly shorter:

```
do_long_running_command
[ $? -eq 0 ] && printf "Command succeeded\n"
```

Wrap up

Thats the basics of shell scripting,

- Include a `#!`
- Always use `env`
- `$?` contains the exit code

Next time

Control flow and more advanced shell scripting for shellscripts.

Bonus puzzle

Why is this the case?

```
[ $? -eq 0 ] # works
[$? -eq 0]   # doesn't work
```

Different shells

(Just use bash unless you care about *extreme* portability in which case use POSIX sh)

Typical Shells

sh	POSIX shell
bash	Bourne Again shell (default on Linux)
zsh	Z Shell (default on Macs), like bash but with more features
ksh	Korne shell (default on BSD)

Other shells

dash	simplified faster bash, used for booting on Linux
Busybox sh	simplified bash you find on embedded systems

Weird shells

fish	More usable shell (but different incompatible syntax)
elvish	Nicer syntax for scripting (but incompatible with POSIX)
nushell	Nicer output (but incompatible, and weird)

Shell Scripting 2

Joseph Hallett

January 12, 2023

University of BRISTOL

Last time

We introduced shell scripting as a tool for automating stuff

- Gave a basic overview of syntax
- Mentioned `env` and `shellcheck`

This time

- More syntax and control flow
- Variables and techniques
  As before I'll try and keep to POSIX shell and mark where things are Bashisms...
  - but some Bash-isms are useful to know

Variables

All programs have variables... Shell languages are no different:

To create a variable:

```
GREETING="Hello World!"
```

(No spaces around the =)

To use a variable

```
echo "${GREETING}"
```

If you want your variable to exist in the programs you start as an *environment variable*:

```
export GREETING
```

To get rid of a variable

```
unset GREETING
```

Well...

Variables in shell languages *tend* to act more like macro variables.

- There's no penalty for using one thats not defined.

```
NAME='Joe'
unset NAME
echo "Hello, '${NAME}'"
```

```
Hello, ''
```

If this bothers you:

```
set -o nounset
echo "${NAME:? variable 1 passed to program}"
```

(There are a *bunch* of these shell parameter expansion tricks beyond `:?` which can do search and replace, string length and various magic...)

Standard variables

`${0}`	Name of the script
`${1}, ${2}, ${3}...`	Arguments passed to your script
`${#}`	The number of arguments passed to your script
`${@} and ${*}`	All the arguments

Control flow

If statements and *for* loops, with *globbing*, are available:

```
# Or [ -x myscript.sh ];
# Or [[ -x myscript.sh ]]; if using Bash
if test -x myscript.sh; then
    ./myscript.sh
fi

for file in *.py; do
    python "${file}"
done
```

Other loops

Well...okay you only have `for` really... but you can do other things with it:

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

1 2 3 4 5

```
seq 5
```

1 2 3 4 5

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

1 2 3 4 5

```
seq -s, 5
```

1,2,3,4,5

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

1 2 3 4 5

Case statements too!

```
3  # Remove everything upto the last / from ${SHELL}
  case "${SHELL##*/}" in
      bash) echo "I'm using bash!" ;;
      zsh)  echo "Ooh fancy a zsh user!" ;;
      fish) echo "Something's fishy!" ;;
      *)    echo "Ooh something else!" ;;
  esac
```

Basename and Dirname

In the previous example I used the "${VAR##*/}" trick to remove everything up to the last /...
Which gives you the name of the file neatly...
...but I have to look this up everytime I use it.
Instead we can use $(basename "${shell}") to get the same info.

```
echo "${SHELL}"
echo "${SHELL##*/}"
echo "$(basename "${SHELL}")"
echo "$(dirname "${SHELL}")"
```

You can even use it to remove *file extensions*:

```
for f in *.jpg; do
  convert "${f}" "$(basename "${f}" .jpg).png"
done
```

Pipelines

As part of shell scripting, its often useful to build commands out of chains of other commands. For example I can use `ps` to list all the processes on my computer and `grep` to search.

- How many processes is *Firefox* using?

```
ps -A | grep -i firefox
```

```
43172   ??   SpU    0:10.69   /usr/local/bin/firefox
59551   ??   Sp     0:00.06   /usr/local/lib/firefox/firefox   -contentproc   -appDir
 7023   ??   SpU    0:06.10   /usr/local/lib/firefox/firefox   -contentproc   {a032331
59478   ??   SpU    0:00.21   /usr/local/lib/firefox/firefox   -contentproc   {3cd651d
47320   ??   SpU    0:00.60   /usr/local/lib/firefox/firefox   -contentproc   {50d5261
26734   ??   SpU    0:00.18   /usr/local/lib/firefox/firefox   -contentproc   {68aa722
  308   ??   SpU    0:00.16   /usr/local/lib/firefox/firefox   -contentproc   {bd6ff5f
42479   ??   SpU    0:00.14   /usr/local/lib/firefox/firefox   -contentproc   {d874750
45572   ??   Rp/2   0:00.00   grep                             -i             firefox
```

Too much info!

Lets use the `awk` command to cut it to just the first and fifth columns!

```
ps -A | grep -i firefox | awk '{print $1, $5}'
```

```
43172   /usr/local/bin/firefox
59551   /usr/local/lib/firefox/firefox
 7023   /usr/local/lib/firefox/firefox
59478   /usr/local/lib/firefox/firefox
47320   /usr/local/lib/firefox/firefox
26734   /usr/local/lib/firefox/firefox
  308   /usr/local/lib/firefox/firefox
42479   /usr/local/lib/firefox/firefox
 5634   grep
```

Why is grep in there?

Oh yes... when we search for *firefox* we create a new process with *firefox* in its commandline. Lets drop the last line

```
ps -A | grep -i firefox | awk '{print $1, $5}' | ghead -n -1
```

```
43172   /usr/local/bin/firefox
59551   /usr/local/lib/firefox/firefox
 7023   /usr/local/lib/firefox/firefox
59478   /usr/local/lib/firefox/firefox
47320   /usr/local/lib/firefox/firefox
26734   /usr/local/lib/firefox/firefox
  308   /usr/local/lib/firefox/firefox
42479   /usr/local/lib/firefox/firefox
```

And really I'd just like a count of the number of processes

```
ps -A | grep -i firefox | awk '{print $1, $5}' | ghead -n -1 | wc -l
8
```

Other piping techniques

- The | pipe copies *standard output* to *standard input*...
- The > pipe copies *standard output* to a named file... (e.g. `ps -A >processes.txt`, see also the `tee` command)
- The >> pipe appends *standard output* to a *named file*...
- The < pipe reads a file *into* standard input... (e.g. `grep firefox <processes.txt`)
- The <<< pipe takes a string and *places it on standard input*
- You can even copy and merge streams if you know their file descriptors (e.g. appending `2>&1` to a command will run it with *standard error* merged into *standard output*)

Wrap up

Go forth and shell script!

What we covered

- Variable expansions
- Common control flow statements
- Different pipe tricks

Build Tools

Joseph Hallett

January 13, 2023

So what's all this about?

An *awful* lot of the things we do with a computer are about *format shifting*
We do this when we compile code:

- `cc -c library.c -o library.o`
- `cc hello.c library.o -o hello`

When we archive files:

- `zip -r coursework.zip coursework`

When we draw figures:

- `dot -Tpdf flowchart.dot -O flowchart.pdf`

Can we automate this?

YES!

We *could* write a shellscript and stick all the tasks in one place...

```
#! /usr/bin/env bash
cc -c library.c -o library.o
cc hello.c library.o -o hello
zip -r coursework.zip coursework
dot -Tpdf flowchart.dot -O flowchart.pdf
```

But can we do better than this?

- Do we really need to recompile the C program if only our flowchart has changed?
- Can we generalise build patterns?

Make

Make in an *ancient* tool for automating builds.

- Developed by *Stuart Feldman* in 1976
- Takes *rules* which tell you how to build files
- Then follows them to build the things you need!

Two main dialects of it (nowadays):

BSD Make More old fashioned, POSIX

GNU Make More featureful, default on Linux

In practice, unless your developing a BSD *every one* uses GNU Make

- If you're on a Mac, or BSD box install GNU Make and try `gmake` if things don't work

Makefiles

Rules for Make are placed into a *Makefile* and look like the following:

```
hello: hello.c library.o
	cc -o hello hello.c library.o

library.o: library.c
	cc -c -o library.o library.c

coursework.zip: coursework
	zip -r coursework.zip coursework

flowchart.pdf: flowchart.dot
	dot -Tpdf flowchart.dot -O flowchart.pdf
```

If you ask make to build `hello` it will figure out what it needs to do:

```
$ make hello
cc -c -o library.o library.c
cc -o hello hello.c library.o
```

Making changes

If you alter files... Make is smart enough to only rerun the steps you need:
For example if you edit `hello.c` and rebuild:

```
$ make hello
cc -o hello hello.c library.o
```

But if you edit `library.c` it can figure out it needs to rebuild *everything*

```
$ make hello
cc -c -o library.o library.c
cc -o hello hello.c library.o
```

Phony targets

As well as rules for how to make files you can have *phony* targets that don't depend on files but just tell make what to do when they're run

Often a `Makefile` will include a phony:

- `all` typically first rule in a file (or marked `.default`): depends on everything you'd like to build
- `clean` deletes all generated files
- `install` installs the program

```
$ make
cc -c -o library.o library.c
cc -o hello.o hello.c library.o
zip -r coursework.zip coursework
dot -Tpdf flowchart.dot -O flowchart.pdf
```

```
.PHONY: all clean

all: hello coursework.zip flowchart.pdf

clean:
        git clean -dfx

hello: hello.c library.o
        cc -o hello hello.c library.o

library.o: library.c
        cc -c -o library.o library.c

coursework.zip: coursework
        zip -r coursework.zip coursework

flowchart.pdf: flowchart.dot
        dot -Tpdf flowchart.dot -O flowchart.pdf
```

Pattern rules

(So far, everything *should* have worked in GNU and BSD Make... here on out we're in GNU land)
What if we wanted to add an extra library to our `hello` programs? We could go and update the `Makefile` but its better to generalise!

```
CC=clang
CFLAGS=-Wall -O3

.PHONY: all clean

all: hello coursework.zip flowchart.pdf
clean:
        git clean -dfx

hello: hello.c library.o extra-library.o

%.o: %.c
        $(CC) $(CFLAGS) -c -o $@ $<

%: %.c
        $(CC) $(CFLAGS) -o $@ $<

%.zip: %
        zip -r $@ $<

%.pdf: %.dot
        dot -Tpdf $< -O $@
```

Implicit pattern rules

Actually because Make is so old, it knows about compiling C (and Fortran/Pascal...) code already:

```
.PHONY: all clean

all: hello coursework.zip flowchart.pdf
clean:
	git clean -dfx

hello: hello.c library.o extra-library.o

%.zip: %
	zip -r $@ $<

%.pdf: %.dot
	dot -Tpdf $< -O $@
```

Lets get even more general!

Suppose we wanted to add more figures... we could add dependencies on `all` to build them or...

```
.PHONY: all clean
figures=$(patsubst .dot,.pdf,$(wildcard *.dot))

all: hello coursework.zip ${figures}
clean:
	git clean -dfx

hello: hello.c library.o extra-library.o

%.zip: %
	zip -r $@ $<

%.pdf: %.dot
	dot -Tpdf $< -O $@
```

Make is crazy powerful

I love Make...

- I abuse it for compiling everything
- For distributing reproducible science studies
- For building and deploying websites

Pattern rules and the advanced stuff is neat...

- ...but if you never use it I won't be offended
- Make is one of those tools that you'll come back to *again and again* over your careers.
- ...and there's a *bunch* of tricks I haven't shown you `;-)`

Go and read the *GNU Make Manual*

- Its pretty good for a technical document

In conclusion

When you get a bit of software... and you find a `Makefile` in there...
Just type `make`!

- (and make sure your projects build in the same way!)

(Actually often you'll have to type `./configure` then `make` for reasons we'll come to *next time*.)

- No I'm not going to teach you *autotools* don't worry!

Buildtools 2: Language specific buildtools

Joseph Hallett

January 13, 2023

Last time...

We talked about *Make* as a tool for compiling code

- We discussed how *Make* could be used to shift documents between different formats

But there is one thing *Make*'s can't do...

Versioning

Modern development makes extensive use of external libraries…
But *Make* is rubbish at dealing with them:

- Doesn't know how to fetch dependencies
- Doesn't track versions beyond *source is newer than object*

LanguageTool is a cool little Java grammar checker:

- How many libraries does *just the core* of the tool make use of?

```
mvn dependency:tree -D outputType=dot | dot -Tpdf
```

This is surely too many?

In the old days...

Traditionally you'd have to go download all the dependencies by hand...

- And then compile and install them
- Very tedious and error prone

So we automated it!

Modern build tooling

(Almost) every language comes with its own library management tooling

- Lets developers specify dependencies
- Tells compiler how to rebuild the project

...which means *for every language you use you need to learn its build tools*...

- Yay?

(Honestly, I still use *Make* but I'm old and cantankerous)

So now we have...

Commonlisp	ASDF and Quicklisp
Go	Gobuild
Haskell	Cabal
Java	Ant, Maven, Gradle...
JavaScript	NPM
Perl	CPAN
Python	Distutils and `requirements.txt`
R	CRAN
Ruby	Gem
Rust	Cargo
LaTeX	CTAN and TeXlive

...and *many* more.

And they're all different

Very little similarity between *any* of them.

- You need to learn the ones you use.
- We'll play in the labs with *Maven* for Java a little bit

Maven Quickstart

```
mkdir /tmp/src
cd /tmp/src
mvn archetype:generate \
    -DgroupId=uk.ac.bristol.cs \
    -DartifactId=hello \
    -DarchetypeArtifactId=maven-archetype-quickstart \
    -DinteractiveMode=false
```

```
INFO Scanning for projects\ldots{}
INFO
INFO ------------------< org.apache.maven:standalone-pom >------------------
INFO Building Maven Stub Project (No POM) 1
INFO --------------------------------[ pom ]---------------------------------
INFO
INFO >>> maven-archetype-plugin:3.2.1:generate (default-cli) > generate-sources @ standalone-
INFO
INFO <<< maven-archetype-plugin:3.2.1:generate (default-cli) < generate-sources @ standalone-
INFO
INFO
INFO --- maven-archetype-plugin:3.2.1:generate (default-cli) @ standalone-pom ---
INFO Generating project in Batch mode
INFO ----------------------------------------------------------------------------
```

...and after spewing all that...

```
find /tmp/src -type f
```

- (”/tmp/src/hello/pom.xml”)
- (”/tmp/src/hello/src/main/java/uk/ac/bristol/cs/App.java”)
- (”/tmp/src/hello/src/test/java/uk/ac/bristol/cs/AppTest.java”)

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>uk.ac.bristol.cs</groupId>
  <artifactId>hello</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>hello</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

And if we try and build...

```
mvn package
[INFO] Scanning for projects\ldots{}
[INFO]
[INFO] ------------------------< uk.ac.bristol.cs:hello >------------------------
[INFO] Building hello 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ hello ---
[WARNING] Using platform encoding (US-ASCII actually) to copy filtered resources, i.e. build is
[INFO] skip non existing resourceDirectory /tmp/src/hello/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ hello ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding US-ASCII, i.e. build is platf
[INFO] Compiling 1 source file to /tmp/src/hello/target/classes
[INFO] -------------------------------------------------------------
[ERROR] COMPILATION ERROR :
[INFO] -------------------------------------------------------------
[ERROR] Source option 5 is no longer supported. Use 7 or later.
[ERROR] Target option 5 is no longer supported. Use 7 or later.
[INFO] 2 errors
```

Lets fix that for it shall we?

(It's either that or installing an ancient Java compiler...)

```
ed /tmp/src/hello/pom.xml <<EOF
10i
  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
  </properties>
.
wq
EOF
```

639 778

And if we try and build, again...

```
mvn package
[INFO] Scanning for projects\ldots{}
[INFO]
[INFO] ------------------------< uk.ac.bristol.cs:hello >------------------------
[INFO] Building hello 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ hello ---
[WARNING] Using platform encoding (US-ASCII actually) to copy filtered resources, i.e. build is
[INFO] skip non existing resourceDirectory /tmp/src/hello/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ hello ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding US-ASCII, i.e. build is platf
[INFO] Compiling 1 source file to /tmp/src/hello/target/classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ hello ---
[WARNING] Using platform encoding (US-ASCII actually) to copy filtered resources, i.e. build is
[INFO] skip non existing resourceDirectory /tmp/src/hello/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ hello ---
```

Does anyone actually know why Java stuff is so ridiculously verbose?

`find /tmp/src -type f`

- ("/tmp/src/hello/pom.xml")
- ("/tmp/src/hello/src/main/java/uk/ac/bristol/cs/App.java")
- ("/tmp/src/hello/src/test/java/uk/ac/bristol/cs/AppTest.java")
- ("/tmp/src/hello/target/maven-status/maven-compiler-plugin/compile/default-compile/createdFiles.lst")
- ("/tmp/src/hello/target/maven-status/maven-compiler-plugin/compile/default-compile/inputFiles.lst")
- ("/tmp/src/hello/target/maven-status/maven-compiler-plugin/testCompile/default-testCompile/createdFiles.lst")
- ("/tmp/src/hello/target/maven-status/maven-compiler-plugin/testCompile/default-testCompile/inputFiles.lst")
- ("/tmp/src/hello/target/classes/uk/ac/bristol/cs/App.class")
- ("/tmp/src/hello/target/test-classes/uk/ac/bristol/cs/AppTest.class")
- ("/tmp/src/hello/target/surefire-reports/uk.ac.bristol.cs.AppTest.txt")
- ("/tmp/src/hello/target/surefire-reports/TEST-uk.ac.bristol.cs.AppTest.xml")
- ("/tmp/src/hello/target/maven-archiver/pom.properties")
- ("/tmp/src/hello/target/hello-1.0-SNAPSHOT.jar")

Other useful commands

`mvn test` run the test suite

`mvn install` install the JAR into your local JAR packages

`mvn clean` delete everything

And if I'm being a bit snarky...

`https://gradle.org` A *better* Java build tool

(That doesn't work everywhere and is much worse than Maven when you try and do more complex things...)

Wrap up

- Language specific build tools exist
- You should probably use them
- (but I still use good ol' make a lot more)

Aside

Sometimes you'll find you pull a project and it uses a certain build system and you just know you're going to have to spend a day fighting it.
...please don't use CMake.

Debugging

Joseph Hallett

January 18, 2023

University of BRISTOL

Whats all this about?

Writing programs is hard

- We should have strategies and *tools* for when things go wrong

Lets point you towards some!

An example program

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(int argc, char *argv[]) {
  char message[128];
  size_t message_len = 256;
  char timestamp[128];
  time_t t;
  struct tm *tmp;
  FILE *file = fopen(argv[1], "a+");

  printf("Type your log: ");
  getline(&message, &message_len, stdin);

  t = time(NULL);
  tmp = localtime(&t);
  strftime(timestamp, 256, "%C", tmp);

  fprintf(file, "%s: %s\n", timestamp, message);
  return 0;
}
```

Lets compile!

```
make journal
cc journal.c -o journal journal.c: In function ‘main’: journal.c:14:11: warning: passing argument 1 of ‘getline’ from incompatible pointer type [-Wincompatible-pointer-types]
In file included from journal.c:1: /usr/include/stdio.h:645:45: note: expected ‘char * restrict’ but argument is of type ‘char (*)[128]’
```

And when we run...

```
./journal <<<"Hello World!"
Segmentation fault (core dumped)
```

Okay, lets try and debug

```
   # gdb ./journal
Reading symbols from ./journal...
(No debugging symbols found in ./journal)
(gdb) run <<<"hello"
Starting program: /home/joseph/Repos/Talks/COMS10012-Software-Tools/Debugging/journal <<<"hello"
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
__vfprintf_internal (s=0x0, format=0x402026 "%s: %s\n", ap=ap@entry=0x7fffffffde50, mode_flags=mode_flags@entry=0) at vfpr
722^^I ORIENT;
(gdb) bt
#0  __vfprintf_internal (s=0x0, format=0x402026 "%s: %s\n",
    ap=ap@entry=0x7fffffffde50, mode_flags=mode_flags@entry=0)
    at vfprintf-internal.c:722
#1  0x00007ffff7e2360a in __fprintf (stream=<optimized out>,
    format=<optimized out>) at fprintf.c:32
#2  0x00000000040125f in main ()
```

Lets make it a *little* easier

```
  cc -Og -g journal.c -o journal
  gdb ./journal
  (gdb) run <<<"hello"
Starting program: /home/joseph/Repos/Talks/COMS10012-Software-Tools/Debugging/journal <<<"hello"
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".

Program received signal SIGSEGV, Segmentation fault.
__memcpy_avx_unaligned_erms () at ../sysdeps/x86_64/multiarch/memmove-vec-unaligned-erms.S:333
Downloading 0.01 MB source file /usr/src/debug/glibc-2.36.9000-19.fc38.x86_64/string/../sysdeps/x86_64/multiarch/memmove-v
333^^I^^Imovl^^I%ecx, -4(%rdi, %rdx)
(gdb) bt
#0  __memcpy_avx_unaligned_erms ()
    at ../sysdeps/x86_64/multiarch/memmove-vec-unaligned-erms.S:333
#1  0x00007ffff7e496ac in __GI___getdelim (
    lineptr=lineptr@entry=0x7fffffffdff0, n=n@entry=0x7fffffffdfe8,
    delimiter=delimiter@entry=10, fp=0x7ffff7fa5aa0 <_IO_2_1_stdin_>)
    at iogetdelim.c:111
#2  0x00007ffff7e237d1 in __getline (lineptr=lineptr@entry=0x7fffffffdff0,
    n=n@entry=0x7fffffffdfe8, stream=<optimized out>) at getline.c:28
#3  0x00000000004011d6 in main (argc=<optimized out>, argv=<optimized out>)
    at journal.c:14
```

Looks like it all went wrong on line 14 of `journal.c`...

```
  (gdb) b journal.c:14
Breakpoint 2 at 0x4011ba: file journal.c, line 14.
(gdb) run <<<"hello"
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/joseph/Repos/Talks/COMS10012-Software-Tools/Debugging/journal <<<"hello"
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".

Breakpoint 2, main (argc=<optimized out>, argv=<optimized out>) at journal.c:14
14^^I getline(&message, &message_len, stdin);
(gdb) inspect message
$3 = "@\000\000\000\000\000\000\000\200", '\000' <repeats 14 times>, "\006\000\000\000\216\000\000\000\f\000\000\000\b
(gdb) inspect message_len
$4 = 256
(gdb) d
Delete all breakpoints? (y or n) y
(gdb)
```

If in doubt... read the manual

In `man 3 getline`:

> *getline() reads an entire line from stream, storing the address of the buffer containing the text into *lineptr. The buffer is null-terminated and includes the newline character, if one was found.*
>
> *If *lineptr is set to NULL before the call, then getline() will allocate a buffer for storing the line. This buffer should be freed by the user program even if getline() failed.*
>
> *Alternatively, before calling getline(), *lineptr can contain a pointer to a malloc(3)-allocated buffer *n bytes in size. If the buffer is not large enough to hold the line, getline() resizes it with realloc(3), updating *lineptr and *n as necessary.*

Well we're passing a *statically* allocated buffer... lets fix that.

A new *example program

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(int argc, char *argv[]) {
  char *message = NULL;
  size_t message_len;
  char timestamp[128];
  time_t t;
  struct tm *tmp;
  FILE *file = fopen(argv[1], "a+");

  printf("Type your log: ");
  getline(&message, &message_len, stdin);

  t = time(NULL);
  tmp = localtime(&t);
  strftime(timestamp, 256, "%C", tmp);

  fprintf(file, "%s: %s\n", timestamp, message);
  return 0;
}
```

```
cc -g -Og journal2.c -o journal2
```

And now when we run...

```
$ ./journal2 <<<"hello"
Segmentation fault (core dumped)

 # gdb ./journal2
(gdb) run <<<"hello"
Starting program: /home/joseph/Repos/Talks/COMS10012-Software-Tools/Debugging/journal2 <<<"hell

Program received signal SIGSEGV, Segmentation fault.
0x00007ffff7e2de82 in __vfprintf_internal () from /lib64/libc.so.6
Missing separate debuginfos, use: dnf debuginfo-install glibc-2.36.9000-19.fc38.x86_64
(gdb) bt
#0  0x00007ffff7e2de82 in __vfprintf_internal () from /lib64/libc.so.6
#1  0x00007ffff7e2360a in fprintf () from /lib64/libc.so.6
#2  0x0000000000401225 in main (argc=<optimized out>, argv=<optimized out>) at journal2.c:20
(gdb)
```

...well, we got further...

We could continue with `gdb`

GDB is an extremely powerful debugging tool

- Its also *really* hard to use
- See *Computer Systems B* next year, or *Systems and Software Security* at Masters level
- If you're on a Mac or BSD box check out `lldb`
- Or for a proper tutorial the documentation it refers you to *every time you open it*.

It is *well worth your time to learn*...

- But *this course* is about *Software Tools* and I want to show you *more* of them

Strace

The `strace` tool lets you trace what systemcalls a program uses

- On OpenBSD see `ktrace` and `kdump`
- On MacOS/FreeBSD see `dtruss` and `dtrace`

Lets run it!

```
$ strace ./journal2 <<<'Hello'
execve("./journal2", ["./journal2"], 0x7ffd3ef71360 /* 36 vars */) = 0
brk(NULL)                               = 0x154f000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc01b1b610) = -1 EINVAL (Invalid argument)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=74509, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 74509, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7ff1ca8ef000
close(3)                                = 0
openat(AT_FDCWD, "/lib64/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P|\2\0\0\0\0\0"..., 832)...
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0"...,
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2232840, ...}, AT_EMPTY_PATH) ...
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f...
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0"...,
mmap(NULL, 1961264, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7ff1ca70e00...
mmap(0x7ff1ca734000, 1409024, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DE...
mmap(0x7ff1ca88c000, 339968, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,...
mmap(0x7ff1ca8df000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DEN...
mmap(0x7ff1ca8e5000, 32048, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANO...
close(3)                                = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f...
arch_prctl(ARCH_SET_FS, 0x7ff1ca8ee640) = 0
set_tid_address(0x7ff1ca8ee910)         = 34069
set_robust_list(0x7ff1ca8ee920, 24)     = 0
rseq(0x7ff1ca8eef60, 0x20, 0, 0x53053053) = 0
mprotect(0x7ff1ca8df000, 16384, PROT_READ) = 0
mprotect(0x403000, 4096, PROT_READ)     = 0
mprotect(0x7ff1ca933000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=9788*1024, rlim_max=RLIM64_INFINITY}...
munmap(0x7ff1ca8ef000, 74509)           = 0
getrandom("\x9e\xe1\xb9\x13\x31\x2c\x9e\xee", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x154f000
brk(0x1570000)                          = 0x1570000
openat(AT_FDCWD, NULL, O_RDWR|O_CREAT|O_APPEND, 0666) = -1 EFAULT (Bad address)
newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x5), ...}, AT_EMPTY_PATH
newfstatat(0, "", {st_mode=S_IFIFO|0600, st_size=0, ...}, AT_EMPTY_PATH) = 0
read(0, "hello\n", 4096)                = 6
openat(AT_FDCWD, "/etc/localtime", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=3664, ...}, AT_EMPTY_PATH) = 0
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=3664, ...}, AT_EMPTY_PATH) = 0
read(3, "TZif2\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\10\0\0\0\10\0\0\0\0"..., 4096) = 366
lseek(3, -2329, SEEK_CUR)               = 1335
read(3, "TZif2\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\10\0\0\0\10\0\0\0\0"..., 4096) = 232
close(3)                                = 0
--- SIGSEGV {si_signo=SIGSEGV, si_code=SEGV_MAPERR, si_addr=0xc0} ---
+++ killed by SIGSEGV (core dumped) +++
Segmentation fault (core dumped)
```

Too much output!

`strace` lets you use regexp to filter what syscalls you look at

- ...or you could just use `grep`...

```
$ strace -e '/open.*' ./journal2 <<<hello
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib64/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, NULL, O_RDWR|O_CREAT|O_APPEND, 0666) = -1 EFAULT (Bad address)
openat(AT_FDCWD, "/etc/localtime", O_RDONLY|O_CLOEXEC) = 3
--- SIGSEGV {si_signo=SIGSEGV, si_code=SEGV_MAPERR, si_addr=0xc0} ---
+++ killed by SIGSEGV (core dumped) +++
Segmentation fault (core dumped)
```

Oh yeah... we forgot an arg

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(int argc, char *argv[]) {
  char *message = NULL;
  size_t message_len;
  char timestamp[128];
  time_t t;
  struct tm *tmp;
  FILE *file = fopen(argv[1], "a+"); /* line 11 */

  printf("Type your log: ");
  getline(&message, &message_len, stdin);

  t = time(NULL);
  tmp = localtime(&t);
  strftime(timestamp, 256, "%C", tmp);

  fprintf(file, "%s: %s\n", timestamp, message); /* line 20 */
  return 0;
}
```

Lets fix that...

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(int argc, char *argv[]) {
  char *message = NULL;
  size_t message_len;
  char timestamp[128];
  time_t t;
  struct tm *tmp;

  if (argc < 2) { printf("Usage %s path/to/log\n", argv[0]); exit(1); };
  FILE *file = fopen(argv[1], "a+"); /* line 11 */

  printf("Type your log: ");
  getline(&message, &message_len, stdin);

  t = time(NULL);
  tmp = localtime(&t);
  strftime(timestamp, 256, "%C", tmp);

  fprintf(file, "%s: %s\n", timestamp, message); /* line 20 */
  return 0;
}
```

Now when we run!

```
./journal3 documents/log.txt <<<hello
Segmentation fault (core dumped)
```

Lets try `ltrace` this time (no equivalent on other platforms)...

- It traces *library* calls

ltrace and a bit more strace

```
$ ltrace ./journal3 documents/log.txt <<<hello
fopen("documents/log.txt", "a+")                                    = nil
printf("Type your log: ")                                           = 15
getline(0x7fffebcc0fc8, 0x7fffebcc0fc0, 0x7f4bfcf40aa0, 0)          = 6
time(nil)                                                           = 1674045150
localtime(0x7fffebcc0f38)                                           = 0x7f4bfcf47640
strftime("20", 256, "%C", 0x7f4bfcf47640)                           = 2
fprintf(nil, "%s: %s\n", "20", "hello\n" <no return ...>
--- SIGSEGV (Segmentation fault) ---
+++ killed by SIGSEGV +++

$ strace -e openat ./journal3 documents/log.txt <<<hello
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib64/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "documents/log.txt", O_RDWR|O_CREAT|O_APPEND, 0666) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/localtime", O_RDONLY|O_CLOEXEC) = 3
--- SIGSEGV {si_signo=SIGSEGV, si_code=SEGV_MAPERR, si_addr=0xc0} ---
+++ killed by SIGSEGV (core dumped) +++
Segmentation fault (core dumped)
```

Lets fix that...

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <errno.h>

int main(int argc, char *argv[]) {
  char *message = NULL;
  size_t message_len;
  char timestamp[128];
  time_t t;
  struct tm *tmp;

  if (argc < 2) { printf("Usage %s path/to/log\n", argv[0]); exit(1); };
  FILE *file = fopen(argv[1], "a+"); /* line 11 */
  if (file == NULL) {
    perror("Failed to open log");
    exit(2);
  }

  printf("Type your log: ");
  getline(&message, &message_len, stdin);

  t = time(NULL);
  tmp = localtime(&t);
  strftime(timestamp, 256, "%C", tmp);

  fprintf(file, "%s: %s\n", timestamp, message); /* line 20 */
  return 0;
}
```

Now when we run...

```
$ ./journal4 <<<hello
Usage ./journal4 path/to/log

$ ./journal4 documents/log.txt <<<hello
Failed to open log: No such file or directory

$ ./journal4 /etc/passwd <<<hello
Failed to open log: Permission denied

$ ./journal4 /dev/stdout
Type your log: hello
20: hello
```

20?!

From `man 3 strftime`:

> *%c The preferred date and time representation for the current locale. (The specific format used in the current locale can be obtained by calling $nl_{langinfo}$(3) with D_{TFMT} as an argument for the %c conversion specification, and with ERA_{DTFMT} for the %Ec conversion specification.) (In the POSIX locale this is equivalent to %a %b %e %H:%M:%S %Y.)*
> *%C The century number (year/100) as a 2-digit integer. (SU) (The %EC conversion specification corresponds to the name of the era.) (Calculated from tm_{year}.)*

Debugging tools can't catch poorly written code!

But other tools can catch things...

Thinking back to when we fixed up `getline`... it said it would allocate the memory for the line

- ...did we ever free it?

```
$ valgrind ./journal4 /dev/stdout <<<hello
==36111== Memcheck, a memory error detector
==36111== Copyright (C) 2002-2022, and GNU GPL'd, by Julian Seward et al.
==36111== Using Valgrind-3.20.0 and LibVEX; rerun with -h for copyright info
==36111== Command: ./journal4 /dev/stdout
==36111==
20: hello

Type your log: ==36111==
==36111== HEAP SUMMARY:
==36111==     in use at exit: 592 bytes in 2 blocks
==36111==   total heap usage: 13 allocs, 11 frees, 13,684 bytes allocated
==36111==
==36111== LEAK SUMMARY:
==36111==    definitely lost: 120 bytes in 1 blocks
==36111==    indirectly lost: 0 bytes in 0 blocks
==36111==      possibly lost: 0 bytes in 0 blocks
==36111==    still reachable: 472 bytes in 1 blocks
==36111==         suppressed: 0 bytes in 0 blocks
```

Wrap up

In this lecture we've cone over the *very basics* of several debugging tools

- `strace`, `ltrace`, `valgrind` and `gdb` will help deal with most of the bugs you encounter

But so will good defensive programming strategies

- *Always* check the return code of functions
- *Always* check assumptions
- *Always* fix your compiler warnings

...actually get more warnings!

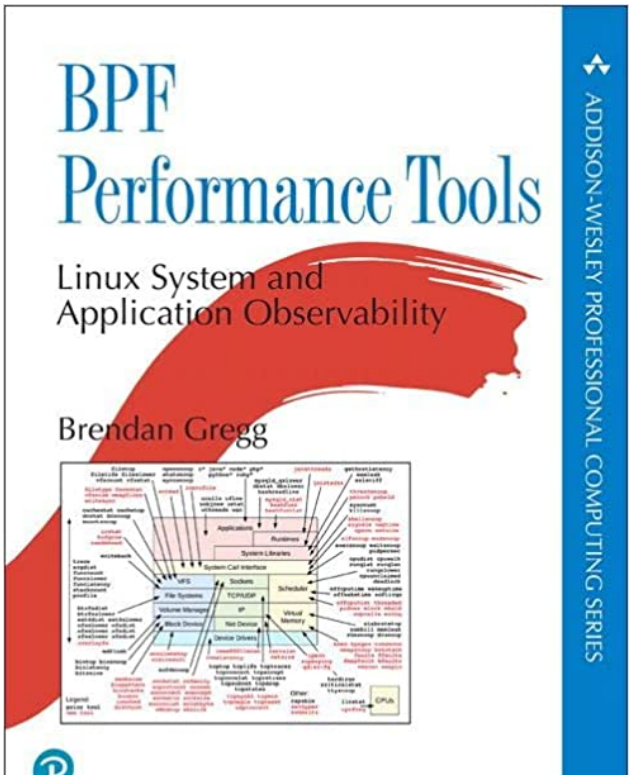
Compiling with the `-Wall -Wextra --std=c11 -pedantic` will make the compiler really picky about your C code...
But there are *other tools* called *linters* that can get even more picky

C/C++	Clang Static Analyser, Rats
Java	FindBugs
Haskell	hlint
Python	pylint, mypy

Other tools for C/C++ can add extra runtime checks

ASan	Address Sanitizer; checks for pointer shenangians
UBSan	Undefined Behaviour Sanitizer; checks for C gotchas

BPF tools



Linux has a (reasonably) new insrumentation framework called *eBPF*

- It lets you get *loads* of detail about what programs are doing
- Highly Linux specific
- I need to learn it :-(

Filesystems, Files and Inodes

Joseph Hallett

January 12, 2023

University of BRISTOL

Whats this about?

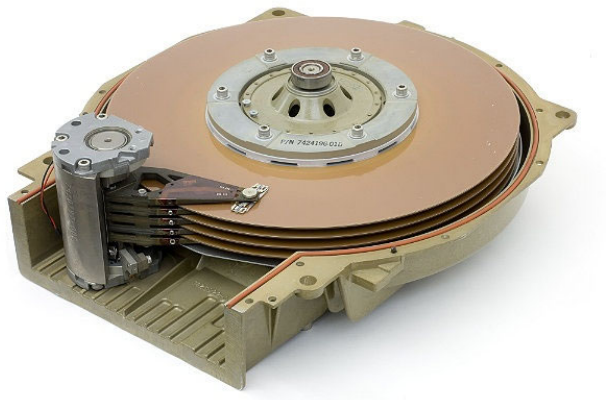
Are harddisks store all our files

- But what *really* is a file and how is the disk storing it?
- ...and how do we gain access to them?
  - (in a low-down way)

Basically we're going to give a quick tour of some low-level gubbins!

- Doing the topic properly could take an entire unit...
- ...maybe a whole degree?

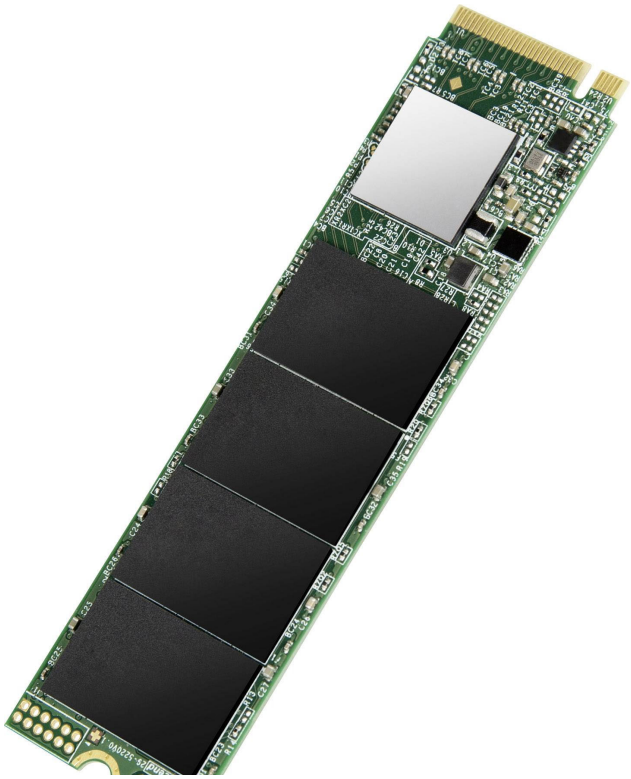
Harddisks?



Heres a harddisk!

- A *cylinder* of platters
- Each *platter* with a *head*
- Each *head* reading from various *cylinders*

Err... they don't look like that?



Heres another harddisk!

- Yes this is what they normally look like now.
- But we still pretend they have cylinders and heads.
- Yes you do occasionally still have to deal with cylinders despite them blatantly not existing.
- Isn't CS fun!?

Luckily for you...

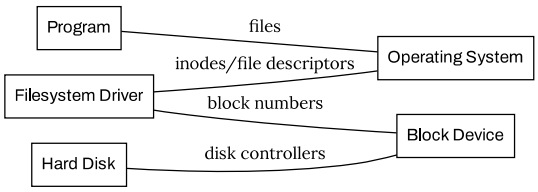
Unless you're writing filesystem drivers you *normally* won't have to deal with these details.

```
ls -l /dev/sd2c
```

brw-r—- 1 root operator 4, 34 Jan 12 08:53 /dev/sd2c
The operating system provides block files

- You can read and write into them like any other file...
- But you *probably* don't want to

Filesystems



As per everything in computer science its a hierarchy

- Different bits talking to different bits
- More and more abstraction

Paths

Filesystems start from a `/`

- The *filesystem root*
- Subsequent directories are separated by further `/`
- Just like URLs... [`https://www.bristol.ac.uk/engineering/departments/computerscience/index.html`] From the root of the webserver at `bris.ac.uk`
  - Go into the folder `engineering`
  - Then the folder `departments`
  - Then the folder `computerscience`
  - Then the thing called `index.html`

Special directories

- `.` the current directory
- `..` the parent directory

```
tree ../
```

```
../
|-- Git-1
|   |-- bristol.png
|   |-- dvcs.pdf
|   |-- git-stage-commit.pdf
|   |-- gitk.png
|   |-- linus.jpg
|   |-- linux.png
|   |-- slides.org
|   |-- slides.pdf
|   |-- slides.tex
|   |-- uk
|   |   `-- ac
|   |       `-- bristol
|   |           `-- cs
|   |               `-- SoftwareTools
|   |                   |-- Hello.class
|   |                   |-- Hello.java
|   |                   |-- Hello2.class
|   |                   `-- Hello2.java
|   |-- vcs.pdf
|   `-- vcs.png
|-- Git-2
|   |-- alices.pdf
|   |-- alicetree.pdf
|   |-- alicetreeprerebase.pdf
```

Interacting with files

High level C API (see `man 3 intro`)

```
#include <stdio.h>


FILE *fopen(const char *pathname, const char *mode);
size_t fread(void *buf, size_t size,
             size_t nmemb, FILE *stream);
size_t fwrite(const void *buf, size_t size,
              size_t nmemb, FILE *stream);
int fclose(FILE *stream);
```

(and all languages provide their own variants)

Low level POSIX API (see `man 2 intro`)

```
#include <fcntl.h>
#include <unistd.h>

int open(const char *path, int flags, ...);
ssize_t read(int d; void *buf; size_t nbytes);

ssize_t write(int d; const void *buf; size_t nbytes);

int close(int d);
```

(and all OSs provide their own variants of these system calls)

So what are these files really?

- The *filesystem* organises files into *paths*...
- The OS lets you interact with files via *file descriptors*
- Your *programming language* gives you a nice API for dealing with them

What actually is a file?

```
ls -i ./
```

```
22852856 bristol.png 22852858 disk.jpg 22852862 fs.pdf 22852863 fslayout.pdf 22852860
slides.org 22852864 slides.pdf 22852861 slides.tex 22852859 ssd.jpg
```

inodes.h

```
struct inode {
  LIST_ENTRY(inode) i_hash;         /* Hash chain */
  structvnode  *i_vnode;            /* Vnode associated with this inode. */
  structufsmount *i_ump;
  u_int32_t i_flag;                 /* flags, see below */
  dev_t  i_dev;                     /* Device associated with the inode. */
  ufsino_t  i_number;               /* The identity of the inode. */
  int       i_effnlink;             /* i_nlink when I/O completes */

  structfs *fs;                     /* FFS */

  struct cluster_info i_ci;
  struct dquot *i_dquot[MAXQUOTAS]; /* Dquot structures. */
  u_quad_t i_modrev;                /* Revision level for NFS lease. */
  struct lockf_state *i_lockf;      /* Byte-level lock state. */
  struct   rrwlock i_lock;          /* Inode lock */

  /* Side effects; used during directory lookup. */
  int32_t  i_count;                 /* Size of free slot in directory. */
  doff_t  i_endoff;                 /* End of useful stuff in directory. */
  doff_t  i_diroff;                 /* Offset in dir, where we found last entry. */
  doff_t  i_offset;                 /* Offset of free space in directory. */
  ufsino_t  i_ino;                  /* Inode number of found directory. */
  u_int32_t i_reclen;               /* Size of found directory entry. */

  /* The on-disk dinode itself. */
  struct ufs2_dinode     *ffs2_din;

  struct inode_vtbl *i_vtbl;
};
```

```
struct ufs2_dinode {
  u_int16_t di_mode;        /*   0: IFMT, permissions; see below. */
  int16_t  di_nlink;        /*   2: File link count. */
  u_int32_t di_uid;         /*   4: File owner. */
  u_int32_t di_gid;         /*   8: File group. */
  u_int32_t di_blksize;     /*  12: Inode blocksize. */
  u_int64_t di_size;        /*  16: File byte count. */
  u_int64_t di_blocks;      /*  24: Bytes actually held. */
  int64_t  di_atime;        /*  32: Last access time. */
  int64_t  di_mtime;        /*  40: Last modified time. */
  int64_t  di_ctime;        /*  48: Last inode change time. */
  int64_t  di_birthtime;    /*  56: Inode creation time. */
  int32_t  di_mtimensec;    /*  64: Last modified time. */
  int32_t  di_atimensec;    /*  68: Last access time. */
  int32_t  di_ctimensec;    /*  72: Last inode change time. */
  int32_t  di_birthnsec;    /*  76: Inode creation time. */
  int32_t  di_gen;          /*  80: Generation number. */
  u_int32_t di_kernflags;   /*  84: Kernel flags. */
  u_int32_t di_flags;       /*  88: Status flags (chflags). */
  int32_t  di_extsize;      /*  92: External attributes block. */
  int64_t  di_extb[NXADDR]; /*  96: External attributes block. */
  int64_t  di_db[NDADDR];   /* 112: Direct disk blocks. */
  int64_t  di_ib[NIADDR];    /* 208: Indirect disk blocks. */
  int64_t  di_spare[3];     /* 232: Reserved; currently unused */
};
```

Simplified version of `/usr/include/ufs/ufs/{,d}inode.h` from OpenBSD

- Other implementations are similar
- Note that the `inode` doesn't have it's name in it...

Directories?

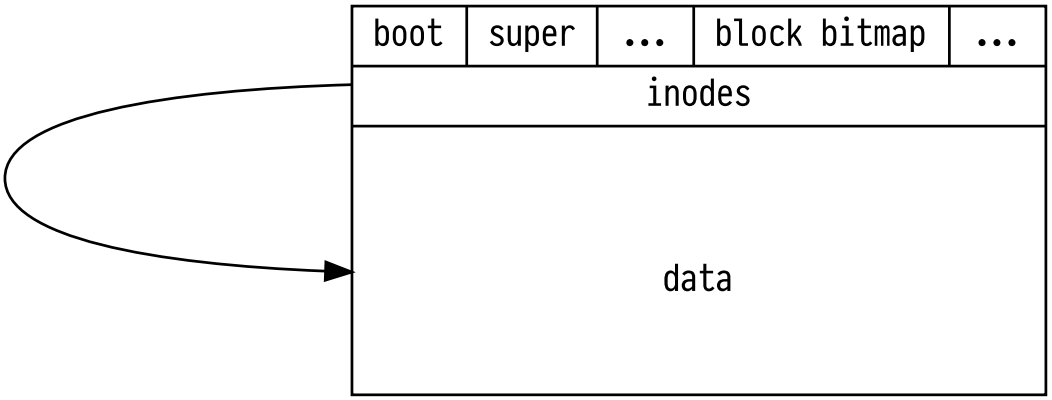
Just *inodes* with a list of *inodes* attached...

```
struct direct {
  u_int32_t d_ino;                /* inode number of entry */
  u_int16_t d_reclen;             /* length of this record */
  u_int8_t  d_type;               /* file type, see below */
  u_int8_t  d_namlen;             /* length of string in d_name */
  char      d_name[MAXNAMLEN + 1]; /* name with length ≤ MAXNAMLEN */
};
```

And for those of us with small screens?

- Just a pointer to a region of memory on disk
- Each file has a unique ID *per filesystem*
- Permissions
- Metadata
- Link count (when 0 safe to reuse)
-

Disk Layout



Links

Two kinds!

Hard Create a file with the *same inode* as another file

- Only works on one filesystem
- Only works for files

Symbolic Create a file that contains as its data an alternative path...

- Works for everything and across filesystems
- Hope tools follow it rather that resolving the actual file
- (Almost all do automatically)

See `man 1 ln`

`ln file link` creates a hard link

`ln -s file link` creates a symbolic link

Other useful stuff

`rm` decrease the link count of an `inode` (if it reaches 0 it'll be deleted)

`rmdir` delete a directory (only works if its empty)

`touch` create or update an inodes file modification times

`readlink` show where a symbolic link links to

`fdisk` create a layout on a disk

`fsck` check the filesystem on a disk (useful after a crash)

`dd` copy raw data to and from a disk

Useful filesystems to dimly recall

FAT what USB drives use

exFAT what bigger USB drives use

ext2 the old Linux filesystem

ext4 the more recent Linux filesystem

Summary

Computers have disks

- Theres an underlying data structure called an *inode*
- The OS will hide all these details away from you
  (*But you probably should know they exist*)

The TTY

Joseph Hallett

January 23, 2023

University of BRISTOL

Whats all this about then?

Bit of computing history

- That appears an awful lot *under the hood*
- You remember how disks no longer have cylinders and platters?
- You remember that I keep using `ed` in these lectures?

A Teletype!

Back before computers had screens...

You'd connect via a *serial* port

- Every character you sent would be typed by the printer
- Including *newlines*, *linefeeds*...
- Little bell for attracting attention if you sent a certain character

You didn't want to have to waste paper and ink (expensive)

- So line editors wouldn't show you what you were editing
- This is the sort of thing that UNIX was written on
- This is the sort of thing we were using 100 years ago

But we're long past that, right?

```
ls /dev/tty*
```

/dev/tty /dev/tty0 /dev/tty1 /dev/tty10 /dev/tty11 /dev/tty12 /dev/tty13 /dev/tty14 /dev/tty15 /dev/tty16 /dev/tty17 /dev/tty18 /dev/tty19 /dev/tty2 /dev/tty20 /dev/tty21 /dev/tty22 /dev/tty23 /dev/tty24 /dev/tty25 /dev/tty26 /dev/tty27 /dev/tty28 /dev/tty29 /dev/tty3 /dev/tty30 /dev/tty31 /dev/tty32 /dev/tty33 /dev/tty34 /dev/tty35 /dev/tty36 /dev/tty37 /dev/tty38 /dev/tty39 /dev/tty4 /dev/tty40 /dev/tty41 /dev/tty42 /dev/tty43 /dev/tty44 /dev/tty45 /dev/tty46 /dev/tty47 /dev/tty48 /dev/tty49 /dev/tty5 /dev/tty50 /dev/tty51 /dev/tty52 /dev/tty53 /dev/tty54 /dev/tty55 /dev/tty56 /dev/tty57 /dev/tty58 /dev/tty59 /dev/tty6 /dev/tty60 /dev/tty61 /dev/tty62 /dev/tty63 /dev/tty7 /dev/tty8 /dev/tty9 /dev/ttyACM0 /dev/ttyACM1 /dev/ttyACM2 /dev/ttyS0 /dev/ttyS1 /dev/ttyS10 /dev/ttyS11 /dev/ttyS12 /dev/ttyS13 /dev/ttyS14 /dev/ttyS15 /dev/ttyS16 /dev/ttyS17 /dev/ttyS18 /dev/ttyS19 /dev/ttyS2 /dev/ttyS20 /dev/ttyS21 /dev/ttyS22 /dev/ttyS23 /dev/ttyS24 /dev/ttyS25 /dev/ttyS26 /dev/ttyS27 /dev/ttyS28 /dev/ttyS29 /dev/ttyS3 /dev/ttyS30 /dev/ttyS31 /dev/ttyS4 /dev/ttyS5 /dev/ttyS6 /dev/ttyS7 /dev/ttyS8 /dev/ttyS9

```
ls /dev/pts/*
```

/dev/pts/0 /dev/pts/2 /dev/pts/ptmx

Pseudo Teletypes

It turns out the *abstraction* that a TTY provides is a useful one

- Device with streams for input and output
- Suitable for keyboard based interaction
- Everything is a file

So UNIX based OSs use it as the fundamental control system

- Connect to terminals with `getty`, `cu`, `screen` or SystemD
- Configure terminals with `stty` and `stdbuf`

Except...

Obviously this *shouldn't work*:

- How do you handle colors?
- How do you handle images?
- How do you handle the mouse?

Some (mostly) standard stuff

The following keys *usually* work

`C-c` interupt
`C-d` end-of-file
`C-l` clear screen
`C-t` display a progress report (rarely works)

If you don't bother to configure `readline`:

`C-a` start of line
`C-e` end of line
`C-k` clear line

(same as *Emacs*... these work in Mac OS everywhere too)

Readline?

Turns out dealing with the forward and backward character IO stuff is a pain
Readline makes it easier by abstracting some of it away

- Gives you command history
- Gives you delete without seeing a bunch of random escape codes
- ...If when using an app you see a bunch of `^[[A` or weird try running the command in `rlwrap`

You can do more with readline

- Have a look at `~/.inputrc` and search for its manual (Linux doesn't always install it)
- Also C bindings worth looking at if you write a lot of CLI apps

```
$if Bash
  "\C-xs": "\C-adoas \C-e"
$endif
```

Escape codes?

So what are these weird `^[[0m` things you see?

- Terminals haven't looked like that teletype since the 1930s

They still *technically* exist

- Receipt printers

...but they're rare.
Most terminals accept there is a screen with more than one row of text displayed at once!

- And all accept a bunch of extra control codes for dealing with multi-line text

Which is standardised right?

...lol.

A little bit?

See `man termcap`

- Then consider a career doing something else

If anything our modern PTY and terminal emulators are *less* featureful than the ones from the 1930s.

- Support for overstruck non-existant
- Advanced formatting pretty much gone
- Loads of support for glitches in particular terminals
- *Meta* key?!
- Non 8-bit bytes!
- Underlining but no overstriking modes!

Don't ever try and do this yourself

If you need to emulate a GUI on the commandline use `ncurses`

If you need to set colors for output use a library

Or in the worst case, use `tput`

`tput setaf 1` make foreground color 1 (red)

`tput setab 2` make the background of text 2 (green)

Make sure to handle the case when you're *not* outputting to a TTY `;-)`
Check `tput colors` to see how many colors your terminal has

- But it lies.

If you really need to do it the escape codes look something like:

- `\e[34mThis text in blue!\e[0m and now back to default`

But if you go beyond the default 8 colors, it rapidly gets weird...

Sometimes you need to emulate a TTY in a TTY

Use `screen` for that

- Or `tmux` if you'd like something less featured but more modern

Conclusion

TTYs still exist.

- Even if they're a pain.
- Mostly things *just work*.
- Expect suffering if you try and go beyond *just working*.

Databases

Joseph Hallett

January 25, 2023

University of BRISTOL

What's all this about?

When you write a program all your data disappears after the program ends

- Unless we save it somewhere

SQL Databases are a sensible choice for where to save your data

- Highly optimized storage of tabular data
- Fast and well understood query language
- Fault tolerant protocols

So what is a database?

Super fancy spreadsheet

- Each database will contain *tables* that store data
- Data in tables can be *queried* using a language called SQL
- Data in tables can be *joined* with data in other tables to answer questions

Designing them so you don't tie yourself in knots is tricky!

So why not just use *Spreadsheets*?

- See *Matt Parker*'s excellent *Stand-up Maths* video: UK Government loses data because of Excel mistake.

https://youtu.be/zUp8pkoeMss

Different types of database

Traditionally, the database would reside on a separate machine

- Space is expensive!
- If you wanted to use the database you had to connect to it

But nowadays space is cheap

- Local per app databases very common

If you need remote data access:

- Use a *server*-style database like *MariaDB* or *MySQL*

Otherwise use a file-style database:

- ...just use *Sqlite*.

To use SQLite

Install the packages however your OS likes to install packages (`apk get` on Alpine)

```
$ sqlite3 database.db
-- Loading resources from /home/joseph/.sqliterc
SQLite version 3.40.1 2022-12-28 14:03:47
Enter ".help" for usage hints.
sqlite> .exit
```

(Or connect via whatever programming language you like)

Seriously, unless you're 100% sure you can't use SQLite: just use SQLite. It's great. Much simpler.

MySQL and MariaDB

If you need a server style install... try MariaDB

History

- It used to be MySQL named after the developer's kid *My* and the languages used to query it SQL
- It was acquired by Oracle...
- A lot of developers don't like Oracle...
- The original developer forked the open source one to make *MariaDB*
- Guess what his other kid's name is?

The command is `mysql` for both

Using MariaDB

```
$ mysql
ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/var/lib/mysql/mysql.so
```

You need to start the server running first or say where to connect to.
On most Linux distros it'll be via SystemD:

```
systemctl start mariadb
systemctl enable mariadb
```

On Alpine Linux it'll be via OpenRC:

```
rc-service mariadb start
```

Security

Once Maria is up and running it'll have some test databases and a `root` user with no password

- Up to you to secure it
- `mysql_secure_installation` can automate most of it...

But if someone is paying you money to do it:

- Set usernames and passwords
- Firewall off ports
- Add logging and intrusion detection
- Backup
- Secure backups
- Have a get out plan...

Otherwise you'll have your database stolen!
(And potentially a *very* large regulatory fine)

Conclusion: Should I use a database?

Am I being paid to store/process this data?

- Yes? Use a database.
- No? Use a spreadsheet (or a database)

Does the data need to be accessed remotely?

- Yes? Use a server-style database (MySQL/MariaDB)
- No? Use a file-style database (SQLite)

Am I just playing with data or is my data tiny (gigabytes in size)?

- Yes? Use a database or plain text data storage (i.e. CSV).

Is my data *really* big (petabytes in size)?

- Yes? Use a NoSQL database (beyond scope of this course)

Does my data contain recursive data structures (i.e. lists of lists of arbitrary length)

- Yes? Use *Prolog* or *Datalog*. (or abuse a database `;-`))

Aside: Pronunciation

How do you say SQL?

- *ess-kew-ell*?
- *sequel*?
- *squirrel*?

Relational Modelling

Joseph Hallett

January 25, 2023

University of BRISTOL

What's all this about then?

Databases let us store data in tables!

- But how do you *structure* your data in a table?
- And can we draw pretty doodles based on them?

Relational modelling

Proviso!

Relational modelling is a tool for thinking about how to decompose relationships between things into tables.

- People get fussy about the syntax

Please don't!

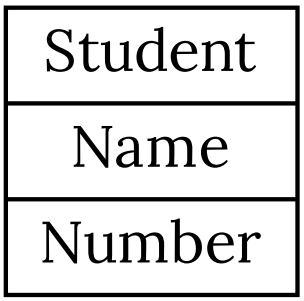
I'll try and show you various syntaxes you may encounter, but its *just a tool*

- Do whatever works for you
- So long as its clear it doesn't matter
- The diagrams are for doodling ideas *not* final implementation

Things are nouns!

Here is a student! Students have a name and a number!

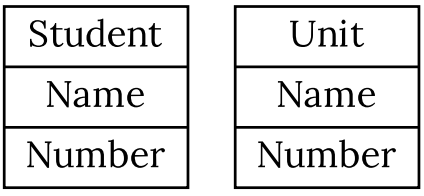
- The student is the *entity*.
- The name and number are the *attributes*.



More things are nouns!

Here is a unit! Units also! have a name and a number!

- The unit is the *entity*.
- The name and number are the *attributes*.



Don’t worry about names

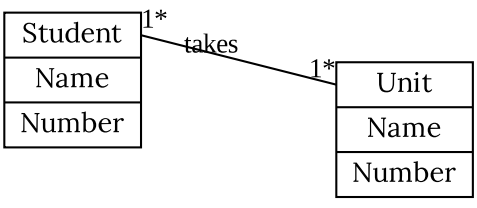
There may be many examples of different *values* that could be examples of units and students... but don’t worry about that.

Student	
Name	Patrick McGoohan
Number	6

Unit	
Name	Software Tools
Number	COMS10012

Nouns can be related!

One student may take many units; and units may have many students



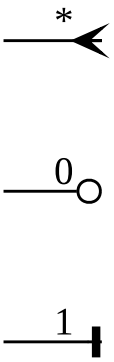
Alternative notation

Some people prefer a graphical notation for entity relationships called *crow's foot*

- I prefer to write it explicitly

Don't get too hung up on notation!

- And use a key if you're ever asked in an exam
- The point is to let *you* doodle notes
- Do whatever makes sense to you or the people you work with



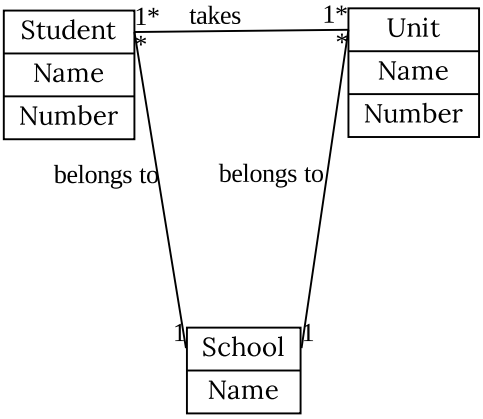
Schools are a thing!

There are things called *schools*:

- Schools have names
- Each unit belongs to *exactly* one school
- Each student belongs to *exactly* one school

Each school can have students and units its responsible for

- But could also be empty!



What should I call a student?

Obviously their name would be *polite*...
...but what will happen if we were to open a class on *Gallifrey*?

All 12!

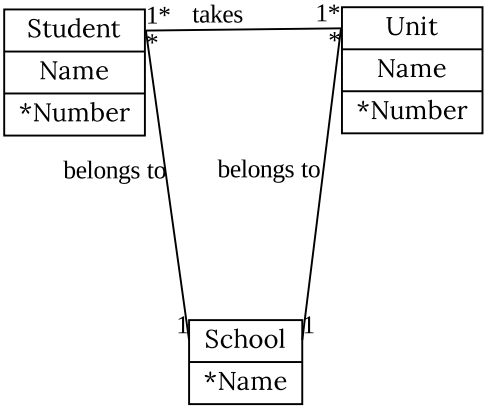
This would rapidly get too confusing for computers!

- (But not for people)

A *key* for an *entity* is the set of attributes needed to uniquely refer to it.

- A *candidate key* is a *minimal* set of attributes needed to uniquely refer to it.
- The *primary key* for an entity is the key we use.

If a key contains multiple attributes its called a *composite key*.
If a key is a meaningless ID column you added just for the sake of having a key its called a *surrogate key*.



When we want to turn it into tables

Every *entity* becomes a *table*

- Each table has a primary key

Every *edge* becomes a table

- Contents of these tables are the *primary keys* of the two items being linked
- Attribute that refers to another key is called a *foreign key*

School Membership

Student	School
6970	School of Computer Science

School Units

Unit	School
COMS10012	School of Computer Science

Student

Name	Number
Joseph Hallett	6970

Unit

Name	Number
Software Tools	COMS10012

School

Name
School of Computer Science

Class Register

Student	Unit
6970	COMS10012

SQL Basics

Joseph Hallett

February 1, 2023

University of BRISTOL

What's all this about?

We've got a database for storing data...

- It'd be nice to be able to acutally use it and make queries!

For that we need SQL:

- Structured Query Language

SQL

Query language for asking questions about databases from 1974

- Standardized in 1986 in the US and 1987 everywhere else
- Still the dominant language for queries today

Not a general purpose programming language

- Not Turing complete
- Weird English-like syntax

Standardized?

You would be so lucky!

- *In theory*, yes
- *In practice*, absolutely not

Every database engine has *small* differences...

- Some have quite big ones too!

Lots have differences in performance

- SQLite is good with strings, most others prefer numbers

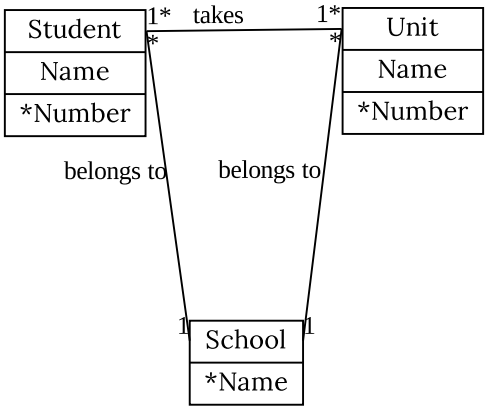
Managing these differences used to be an entire degree/job in its own right!

- Now we just manage databases badly!

I'll try and stick to SQLite's syntax...

CREATE TABLE

In the last lecture we had the following Entity relationship diagram:




```
CREATE TABLE IF NOT EXISTS student (
  name TEXT NOT NULL,
  number TEXT NOT NULL,
  PRIMARY KEY (number));

CREATE TABLE IF NOT EXISTS unit (
  name TEXT NOT NULL,
  number TEXT NOT NULL,
  PRIMARY KEY (number));

CREATE TABLE IF NOT EXISTS school (
  name TEXT NOT NULL,
  PRIMARY KEY (name));

CREATE TABLE IF NOT EXISTS class_register (
  student TEXT NOT NULL,
  unit TEXT NOT NULL,
  FOREIGN KEY (student) REFERENCES student(number),
  FOREIGN KEY (unit) REFERENCES unit(name),
  PRIMARY KEY (student, unit));
```

Lets build it in SQL

DROP TABLE

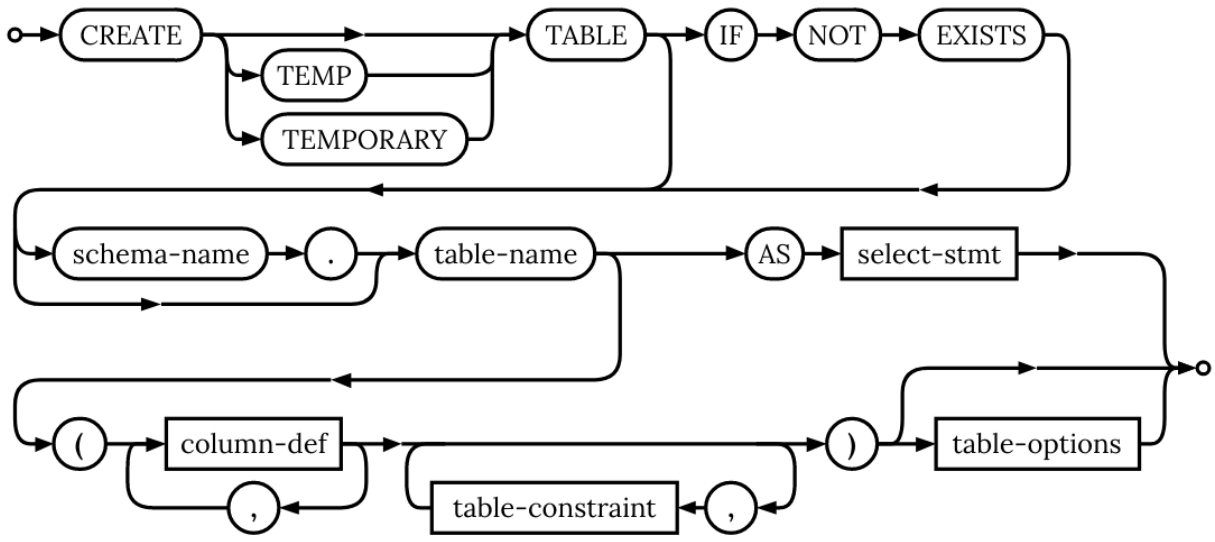
What about if we want to delete them?

```
DROP TABLE IF EXISTS class_register;
DROP TABLE IF EXISTS student;
DROP TABLE IF EXISTS unit;
DROP TABLE IF EXISTS school;
```

Syntax, syntax, syntax

If you go on the *SQLite* documentation page...

- ...you can find syntax diagrams for all of SQL!
- `https://www.sqlite.org/lang_createtable.html`



Types

When creating the fields in our database we made them all of type `TEXT`...

- What other types exist?

`INTEGER`	whole numbers
`REAL`	lossy decimals
`BLOB`	binary data (images/audio/files...)
`VARCHAR(10)`	a string of 10 characters
`TEXT`	any old text
`BOOLEAN`	True or false
`DATE`	Today
`DATETIME`	Today at 2pm

But really types

Databases sometimes *simplify* these types

- SQLite makes the following tweaks...

`INTEGER`	whole numbers
`REAL`	lossy decimals
`BLOB`	binary data (images/audio/files...)
`VARCHAR(10)`	*actually* `TEXT`
`TEXT`	any old text
`BOOLEAN`	*actually* `INTEGER`
`DATE`	*actually* `TEXT`
`DATETIME`	*actually* `TEXT`

(others may exist... *read the manual!*)

Table constraints

In the earlier examples we marked some columns as `NOT NULL`

- Others as `PRIMARY KEY` and others as `FOREIGN KEY`...
- ...what other constraints have we got

...but SQLite won't actually enforce any of these types or constraints unless you ask it to `:-(`

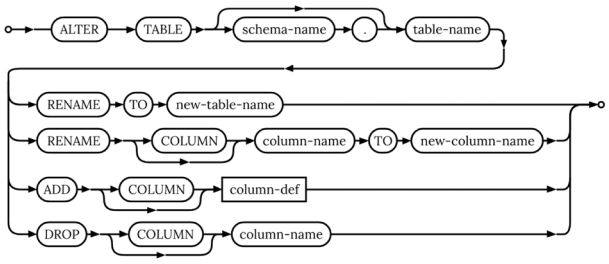
- Check out the `STRICT` keyword when creating the table.

`NOT NULL`	can't be `NULL`
`UNIQUE`	can't be the same as another row
`CHECK`	arbitrary checking (including it conforms to a regular expression)
`PRIMARY KEY`	unique, not `NULL` and (potentially) autogenerated
`FOREIGN KEY`	(IGNORED BY MARIADB) other key must exist

Can I add constraints later?

Yes with the `ALTER TABLE` statement

- But often easiest just to save the table somewhere else
- Drop the table
- Reimport it



INSERT INTO

What about if we want to add data to a table?

```
INSERT INTO unit(name, number)
VALUES ("Software Tools", "COMS100012");
```

So far

We've introduced how to:

- `CREATE TABLE`
- `DROP TABLE`
- `INSERT INTO`

Next step: querying data!

I'm going to use a database from an old iTunes library for demo purposes

- Chinook database

SELECT

Basic command for selecting rows from a table is SELECT

```
SELECT * FROM album
LIMIT 5;
```

AlbumId	Title	ArtistId
1	For Those About To Rock We Salute You	1
2	Balls to the Wall	2
3	Restless and Wild	2
4	Let There Be Rock	1
5	Big Ones	3

```
SELECT * FROM artist
LIMIT 5;
```

ArtistId	Name
1	AC/DC
2	Accept
3	Aerosmith
4	Alanis Morissette
5	Alice In Chains

JOIN

Ideally we'd like those two tables combined into one...

```
SELECT *
FROM album
JOIN artist
ON album.artistid = artist.artistid
LIMIT 5;
```

AlbumId	Title	ArtistId	ArtistId	Name
1	For Those About To Rock We Salute You	1	1	AC/DC
2	Balls to the Wall	2	2	Accept
3	Restless and Wild	2	2	Accept
4	Let There Be Rock	1	1	AC/DC
5	Big Ones	3	3	Aerosmith

Reducing the columns...

Clearly there are too many columns here... lets only select the ones we need

```
SELECT album.title, artist.name
FROM album
JOIN artist
ON album.artistid = artist.artistid
LIMIT 5;
```

Title	Name
For Those About To Rock We Salute You	AC/DC
Balls to the Wall	Accept
Restless and Wild	Accept
Let There Be Rock	AC/DC
Big Ones	Aerosmith

Renaming columns

Title and *Name* aren't particularly meaningful without context

- Lets name them something sensible

```
SELECT album.title AS album,
       artist.name AS artist
FROM album
JOIN artist
ON album.artistid = artist.artistid
LIMIT 5;
```

album	artist
For Those About To Rock We Salute You	AC/DC
Balls to the Wall	Accept
Restless and Wild	Accept
Let There Be Rock	AC/DC
Big Ones	Aerosmith

I'm feeling rocky

I want to listen to something a bit rocky...

- Lets filter all the albums to the ones that have *Rock* in the title

```
SELECT album.title AS album,
       artist.name AS artist
FROM album
JOIN artist
ON album.artistid = artist.artistid
WHERE album LIKE '%Rock%'
LIMIT 5;
```

album	artist
For Those About To Rock We Salute You	AC/DC
Let There Be Rock	AC/DC
Deep Purple In Rock	Deep Purple
Rock In Rio [CD1]	Iron Maiden
Rock In Rio [CD2]	Iron Maiden

Who rocks?

So who has put out an album with *Rock* in it?

```
SELECT artist.name AS artist
FROM album
JOIN artist
ON album.artistid = artist.artistid
WHERE album.title LIKE '%Rock%'
LIMIT 5;
```

artist
AC/DC
AC/DC
Deep Purple
Iron Maiden
Iron Maiden

```
SELECT DISTINCT artist.name AS artist
FROM album
JOIN artist
ON album.artistid = artist.artistid
WHERE album.title LIKE '%Rock%'
LIMIT 5;
```

artist
AC/DC
Deep Purple
Iron Maiden
The Cult
The Rolling Stones

How many *rock* albums has each artist put out?

Lets group by artist and count the albums!

```
SELECT artist.name AS artist,
       COUNT(album.title) as albums
FROM album
JOIN artist
ON album.artistid = artist.artistid
WHERE album.title LIKE '%Rock%'
GROUP BY artist
LIMIT 5;
```

artist	albums
AC/DC	2
Deep Purple	1
Iron Maiden	2
The Cult	1
The Rolling Stones	1

Really we want this list ordered...

Lets group by artist and count the albums...

- And order it by album count!

```
SELECT artist.name AS artist,
       COUNT(album.title) as albums
FROM album
JOIN artist
ON album.artistid = artist.artistid
WHERE album.title LIKE '%Rock%'
GROUP BY artist
ORDER BY albums DESC
LIMIT 5;
```

artist	albums
Iron Maiden	2
AC/DC	2
The Rolling Stones	1
The Cult	1
Deep Purple	1

Conclusions

So thats the basics of SQL!

- You can do a *bunch* more things with SQL `SELECT` statements...
- ...you can pick them up as you write queries.
- ...most SQL engines have a bunch more counting and query functions too

Go read the documentation!

Database Normal Forms

Joseph Hallett

February 1, 2023

University of BRISTOL

What's all this about?

Database theory!

- So far we've discussed how to doodle database designs...
- We've discussed how to create tables in SQL

This time:

How do we design tables that are easy to use?

Lets start with our records database again...

We could store our data as follows:

Artist	Albums
The Beatles	Yellow Submarine, White Album, Rubber Soul
Milk Can	Make It Sweet
Dresden Dolls	Yes Virginia, No Virginia, The Dresden Dolls

Please, no.

This is a *terrible* idea

- Yes we have one big table which seems neater
- But its much harder to do anything actually with

For example:

- How many albums does each artist have?
- Change all of *Prince*'s albums after 1993 to being by a *Love Symbol*
- How many artists have an album with the same name?

Normal forms

Normal forms prevent this sort of insanity

- Using them requires discipline, and rememebering rules...
- But is worth it for your sanity in the short to medium term

First Normal Form

Each column shall contain *one* (and only one) value

Each row says describes *multiple* albums per artist...

Artist	Albums
The Beatles	Yellow Submarine, White Album, Rubber Soul
Milk Can	Make It Sweet
Dresden Dolls	Yes Virginia, No Virginia, The Dresden Dolls

First Normal Form

Lets fix that...

Artist	Album
The Beatles	Yellow Submarine
The Beatles	White Album
The Beatles	Rubber Soul
Milk Can	Make It Sweet
Dresden Dolls	Yes Virginia
Dresden Dolls	No Virginia
Dresden Dolls	The Dresden Dolls

Lets add some more data to our table

Artist	Album	Year	Prime Minister
The Beatles	Yellow Submarine	1969	Harold Wilson
The Beatles	White Album	1968	Harold Wilson
The Beatles	Rubber Soul	1965	Harold Wilson
Milk Can	Make It Sweet	1999	Tony Blair
Dresden Dolls	Yes Virginia	2006	Tony Blair
Dresden Dolls	No Virginia	2008	Gordon Brown
Dresden Dolls	The Dresden Dolls	2003	Tony Blair

Second Normal Form

Every non-key attributue is fully dependent on the key

In this case the key is *Artist*, *Album*

- And arguably *year* too if you're gonna pull a Taylor Swift and rerelease all your albums...

Is *Prime Minister* dependent on the key?

- No. Put it in a different table.

Now it looks like

Artist	Album	Year
The Beatles	Yellow Submarine	1969
The Beatles	White Album	1968
The Beatles	Rubber Soul	1965
Milk Can	Make It Sweet	1999
Dresden Dolls	Yes Virginia	2006
Dresden Dolls	No Virginia	2008
Dresden Dolls	The Dresden Dolls	2003

Year	Prime Minister
1969	Harold Wilson
1968	Harold Wilson
1965	Harold Wilson
1999	Tony Blair
2006	Tony Blair
2008	Gordon Brown
2003	Tony Blair

Third Normal Form

> *Every non-key attribute must provide a fact about the key, the whole key and nothing but the key; so help me Codd.*

Lets add some extra information to our table of Prime Ministers...

Year	Prime Minister	Birthday
1969	Harold Wilson	1916-03-11
1968	Harold Wilson	1916-03-11
1965	Harold Wilson	1916-03-11
1999	Tony Blair	1953-05-06
2003	Tony Blair	1953-05-06
2006	Tony Blair	1953-05-06
2008	Gordon Brown	1951-02-20

Our key is (*Year*, *Prime Minister*); Birthday depends on *Prime Minister*.

- So every non-key depends on the key...
- So 2NF

But not 3NF as Birthday *doesn't* tell you a fact about the *whole key*... just the Prime Minister.

So split it up!

Year	Prime Minister
1969	Harold Wilson
1968	Harold Wilson
1965	Harold Wilson
1999	Tony Blair
2003	Tony Blair
2006	Tony Blair
2008	Gordon Brown

Prime Minister	Birthday
Harold Wilson	1916-03-11
Tony Blair	1953-05-06
Gordon Brown	1951-02-20

Why is this better?

- Now if we need to alter the birthday of a PM (or any other fact about that key)...
- ...then we only need to alter it in one place.

Other normal forms...

Boyce-Codd Normal Form

A slightly stronger form of 3NF...

- Sometimes called 3.5th Normal Form

Every possible *candidate key* for a table is also in 3NF.

- Split a 3NF table into tables with single candidate keys to get 3.5NF.

4th Normal Form

If multiple attributes in a table depend on the same key,

- Then those attributes should be dependant too
- Otherwise split them into separate tables...

5th Normal Form

It's in 4th normal form and you can't split it into more separate tables.

This is all getting a bit mathsy...

You can look up formal definitions for each of the normal forms

- (and you should)

But so long as you keep things *as separate as possible*, you'll usually hit at least 3NF by accident.

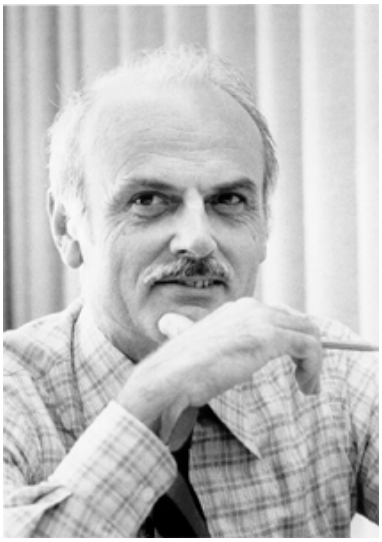
- ...and practically speaking your probably good then
- Getting it to 5NF *does* make things more flexible in the long run...
- But a 3.5NF database is often *good enough*.

Ultimately design is subjective (somewhat).

- ...but mathematical proof of flexibility is good right?

In conclusion

Every non-key attribute must provide a fact about the key, the whole key and nothing but the key; so help me Codd.



Ted Codd