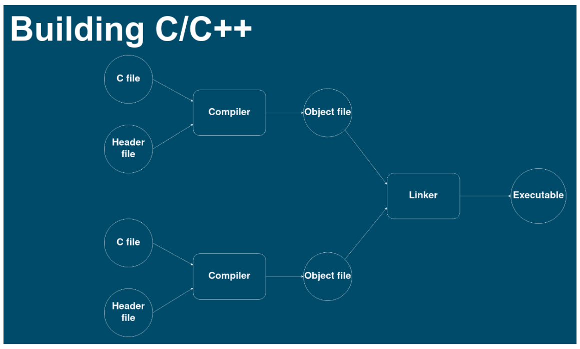
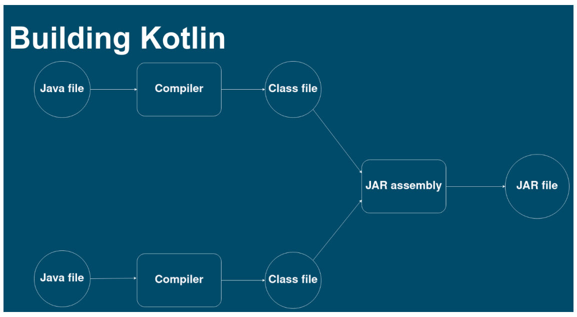
04. Build Systems and Continuous Integration





Building large software systems

You're probably used to having a button on your IDE that can trigger builds for you, but this hides a lot of complexity. How will this process scale if we have thousands of files to compile? Or millions of lines of code?

The process is generally automated using a piece of software called a **build system**.

Common build systems

CMake

Maven™

Gradle

Bazel

Building with make

GNU make is a widely-used build system for C/C++.

GNU make

Commonly (and widely) used on Unix systems. It is very low-level, but pioneered concepts used by almost all build systems. Mostly used for C/C++, but is technically language neutral. Builds are specified by a **Makefile** in the root of the repository.

Offers incremental compilation (after compilation and file modification, it only recompiles the modified files).

```
# Simple Makefile

prog: hello.o mult.o
    gcc -o prog hello.o mult.o

hello.o: hello.c
    gcc -c -o hello.o hello.c

mult.o: mult.c
    gcc -c -o mult.o mult.c
```

```
# Let the user change the build process

CC = gcc
PROG_NAME ?= prog

$(PROG_NAME): hello.o mult.o
    $(CC) -o $(PROG_NAME) hello.o mult.o

hello.o: hello.c
    $(CC) -o hello.c -c hello.c

mult.o: mult.c
    $(CC) -o mult.c -c mult.c
```

```
# Remove redundancy

CC = gcc
PROG_NAME ?= prog
```

```
$(PROG_NAME): hello.o mult.o
	$(CC) -o $(PROG_NAME) hello.o mult.o

%.o: %.c
	$(CC) -o $@ -c $^
```

```
# Add phony targets

CC = gcc
PROG_NAME ?= prog

$(PROG_NAME): hello.o mult.o
	$(CC) -o $(PROG_NAME) hello.o mult.o

%.o: %.c
	$(CC) -o $@ -c $^

.PHONY: clean

clean:
	rm -f *.o
	rm -f $(PROG_NAME)

.PHONY: install
install: $(PROG_NAME)
	install -m 655 -o root $(PROG_NAME) /usr/bin
```

```
# Generate depfiles (useful for debugging)

CC = gcc
PROG_NAME ?= prog

$(PROG_NAME): hello.o mult.o
	$(CC) -o $(PROG_NAME) hello.o mult.o

%.o: %.c
	$(CC) -MD -MF $(subst .c,.d,$^) -o $@ -c $^
```

```
.PHONY: clean

clean:
    rm -f *.o *.d
    rm -f $(PROG_NAME)

.PHONY: install
install: $(PROG_NAME)
    install -m 655 -o root $(PROG_NAME) /usr/bin

# install commands moves the executable to a directory in the path
```

GNU make represents builds as a graph structure. Performs incremental builds (only re-runs tasks if the inputs have changed).

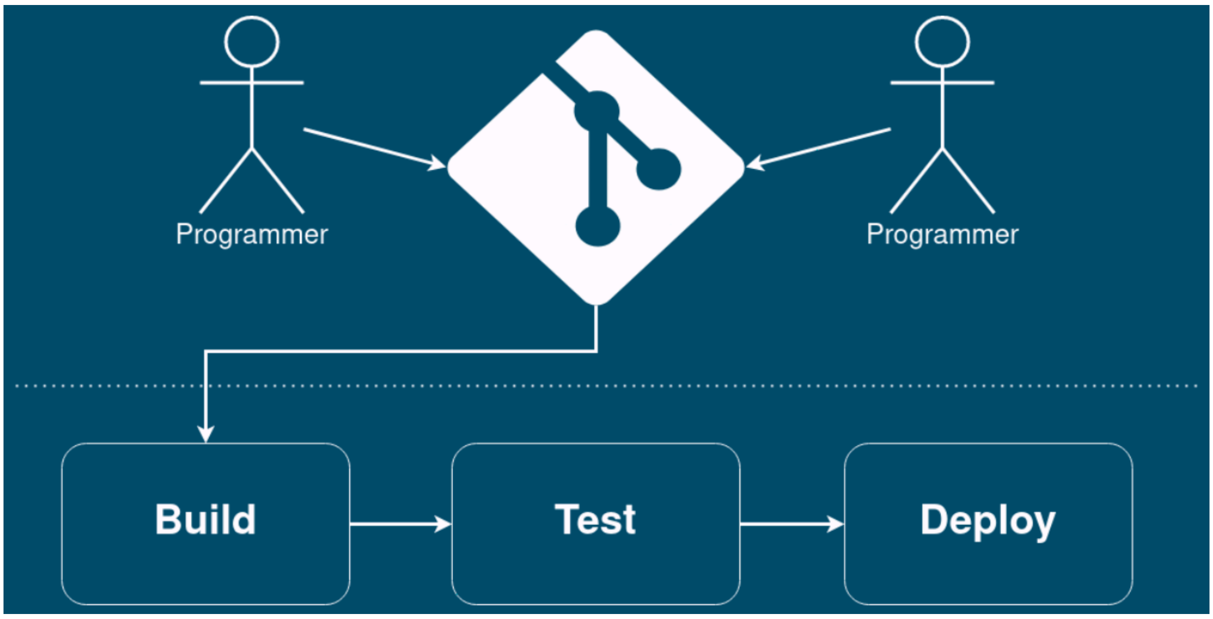
Phony rules are rules which don’t produce an output file. Conventionally, you should create **install** and **clean** phony rules.

To parallelize make build:

```
make -j2 # 2 is the number of cores to use
```

Continuous integration

Run builds or similar automated tasks every time there is a commit.



We'd like to have builds performed on our source code frequently (maybe daily, hourly or even per commit).

When the build completes, we'd like to run tests to ensure that everything works correctly and that the code changes haven't broken anything. This process is called **continuous integration**.

Jenkins

It is a dedicated continuous integration tool written in Java. It is intended to be run on a remote server and provides a web-based interface. It is controlled using a script called a Jenkinsfile.

Jenkins Pipelines

A Jenkinsfile defines a set of tasks called Pipeline. Typically, each task will implement one of the steps from the CI cycle: build, test, deploy.

```
stages {
    stage('Build programme') {
        steps {
            sh 'make -j4'
        }
    }

    stage('Test programme') {
        steps {
            sh 'make run_tests'
        }
    }
}
```