Revision

Dr. Gareth Roy (x6439)
gareth.roy@glasgow.ac.uk

- The Filesystem
- Processes
- Bash Scripting
- Compilation
- Makefiles
- Git
- GDB

- The Filesystem
- Processes
- Bash Scripting
- Compilation
- Makefiles
- Git
- GDB

The Filesystem

- You should be able to:
  - Understand the difference between relative and absolute paths
  - Know what a hidden file is, and give examples of some.
- Navigate and explore the filesystem using commands such as:

```
pwd, tree, ls, ls -l, ls -a, cd, pushd, popd, find, locate, whereas
```

- Modify file permissions with chmod (see next slide)
- Create files and directories:

```
touch, mkdir, mkdir -p
```

- Move and copy files:

```
mv, cp, cp -r
```

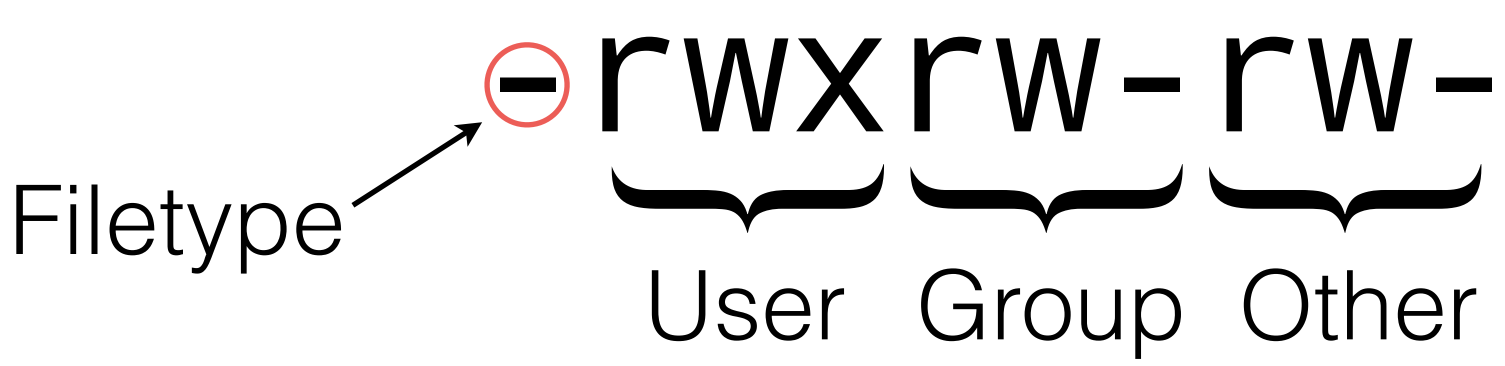
- Delete files:

```
rm, rm -r, rm -f
```

- Explore the contents of files with:

```
file, cat, less, more, head, tail, grep, wc, sort
```

Permissions



- First part is filetype
  - **d** - directory
  - **l** - link
  - **-** - normal file
- Next is permissions, split into three parts
  - **User** permissions
  - **Group** permissions
  - **Other** permissions (everyone on the system)
- Permissions are
  - **r** - read access
  - **w** - write access
  - **x** - execute

```
chmod [who][op][what] filename
```

- Can change permissions of a file using the **chmod** command
- who can be:
  - **u** - user permissions
  - **g** - group permissions
  - **o** - other permissions
  - **a** - all permissions
- op can be:
  - **+** - grant permissions
  - **-** - remove permissions
- what is one of the three permission types (**r,w,x**)

- The Filesystem
- Processes
- Bash Scripting
- Compilation
- Makefiles
- Git
- GDB

Processes

- You should be able to:
- Explain what an environment variable is, how to set one and how to get it's contents:

```
MYVAR='somevalue'; echo ${MYVAR}
```

- Understand what important system variables do:

```
$PATH, $HOME, $PWD, $USER, $LD_LIBRARY_PATH
```

- How to set persistent variables in .bash_profile and .bashrc
- How to monitor processes:

```
pstree, ps, ps -elf, ps -auxwf, top
```

- How to work with processes and basic job control:

```
&, fg, bg, jobs, ctrl-c, ctrl-z, kill, kill -9, pidof
```

- How to work with IO Streams:

```
stdin, stdout, stderr
```

- Redirection and Pipes:

```
<, >, >>, 2>&1, |
```

- The Filesystem
- Processes
- Bash Scripting
- Compilation
- Makefiles
- Git
- GDB

Bash Scripting

- You should be able to:
- Make a simple script and know how to run it using source and by making it executable (chmod):

```
#!/bin/bash
echo “Hello, World!”
```

- Know the difference between single quotes and double quotes and why I needed quotes above (c.f. Bash word splitting).
- Understand and be able to write Bash ranges and lists

```
{1..3}, {a..z}, `seq 1 10`
```

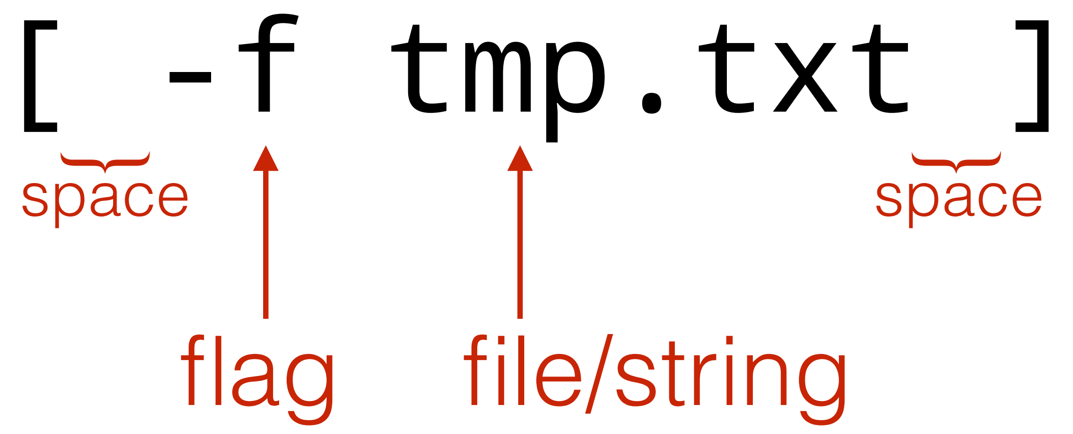
- Know how to use variables in scripts:

```
CALCFUNC=${HOME}/bin/thermCoeff
echo ${CALCFUNC}
```

- Know basic control flow constructions:

```
if [ “${CALCFUNC}” = “” ]; then
   echo “${CALCFUNC} is not set”
fi

while [ -e lock file ]; do
   echo “Cannot start - rm lock file”
   sleep 10
done

for ITEM in $LIST; do
   echo $ITEM
done
```

Bash Conditionals



```
[ -f tmp.txt ]
```

String Comparision	Result
string 1 == string 2	True if the strings are equal
string 1 != string 2	True if the strings are different
-n string	True if the string is not null
-z string	True if the string is null

File Conditionals	Result
-d file	True if file is a directory
-e file	True if file exists
-f file	True if the file is regular
-r file	True if file is readable
-s file	True if file has nonzero size
-w file	True if file is writeable
-x file	True if file is executable

Arithmetic Comparision	Result
exp 1 -eq exp 2	True if both are equal
exp 1 -ne exp 2	True if both are different
exp 1 -gt exp 2	True if exp1 is greater than exp2
exp 1 -ge exp 2	True if exp1 is greater than or equal to exp2
exp 1 -lt exp 2	True if exp1 is less than exp2
exp 1 -le exp 2	True if exp1 is less than or equal to exp2
! exp	Invertes exp, true if exp is false. False if exp is true

Bash Scripting

- Know how to run external commands and store output in variables.

```
VAR=`ls`
VAR=$(wc bob.txt)
```

- Know about exit statuses:

```
exit 0
exit 1
```

- Understand the meanings an uses for special variables:

```
$$, $0, $1…, $#, $?, $@
```

- How to get user input, and read a file line by line:

```
read, read -p,
cat some file | while read LINE; do echo $LINE; done
```

- How to declare and call a Bash function:

```
function myfunc {
    echo “Hello, $1!!”
}
myfunc “Bob”
```

- How to pass arguments to a Bash function and access them using:

```
$@, $#, $1…, shift
```

- How to debug a Bash file using:

```
set -x, set -n, set-v
```

- The Filesystem
- Processes
- Bash Scripting
- Compilation
- Makefiles
- Git
- GDB

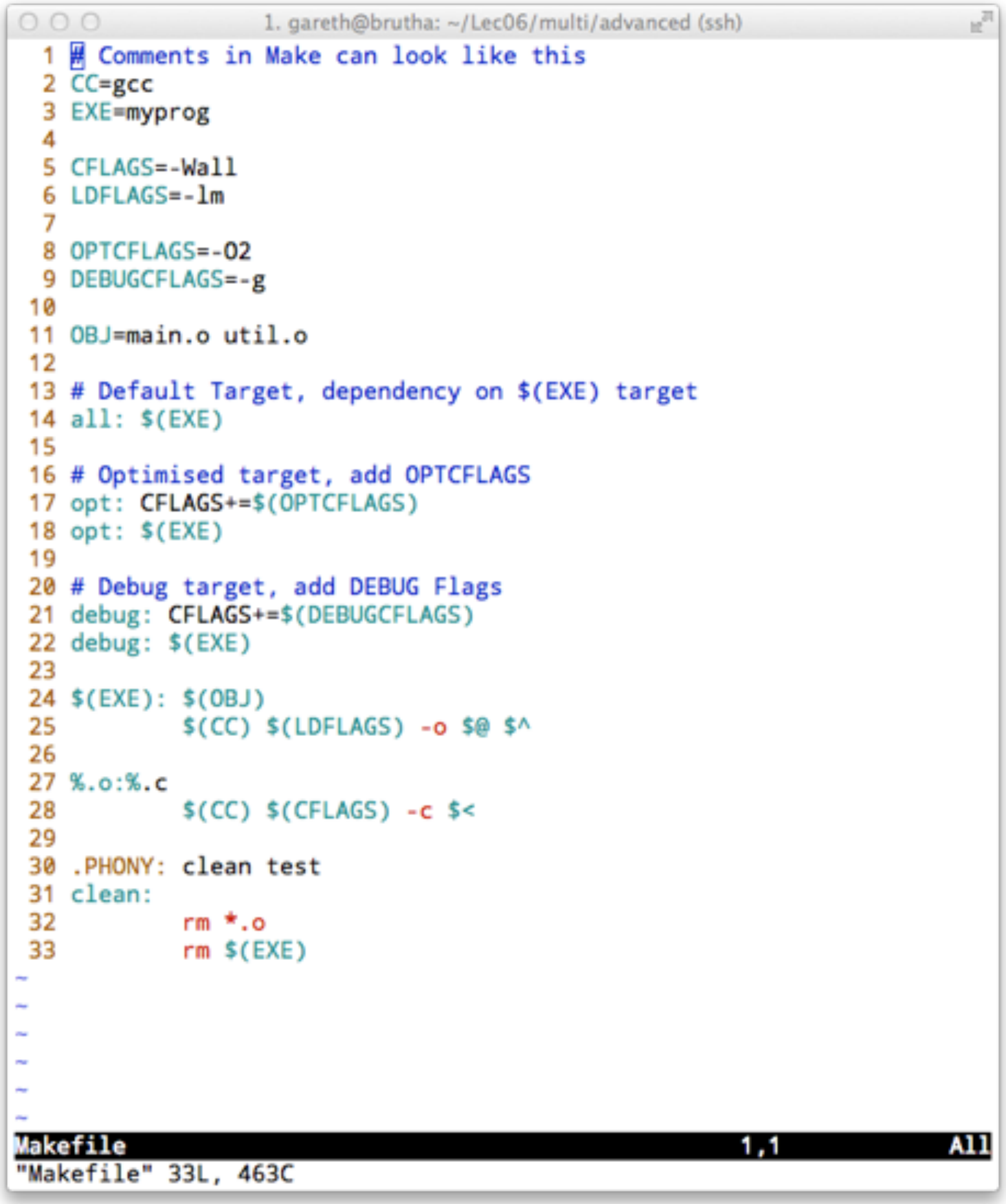
Compilation

- You should be able to:
- Compile C source code to create an executable.
- Understand what the pre-processor, compiler and linker do and how the compilation process works.
- How to use gcc to carry out a one line compilation:
    - `gcc -lm -o myprog *.c`
- How to use gcc to carry out an incremental build:
    - `gcc -c main.c`
    - `gcc -c util.c`
    - `gcc -lm -o myprog main.o util.o`
- Understand what libraries are and how to include them via the gcc compilation line.
- Understand the difference between a source file and a header file.
- The tools that can be used to explore the steps in the build process:
    - `nm, ldd, hexdump`

- The Filesystem
- Processes
- Bash Scripting
- Compilation
- Makefiles
- Git
- GDB

Makefiles

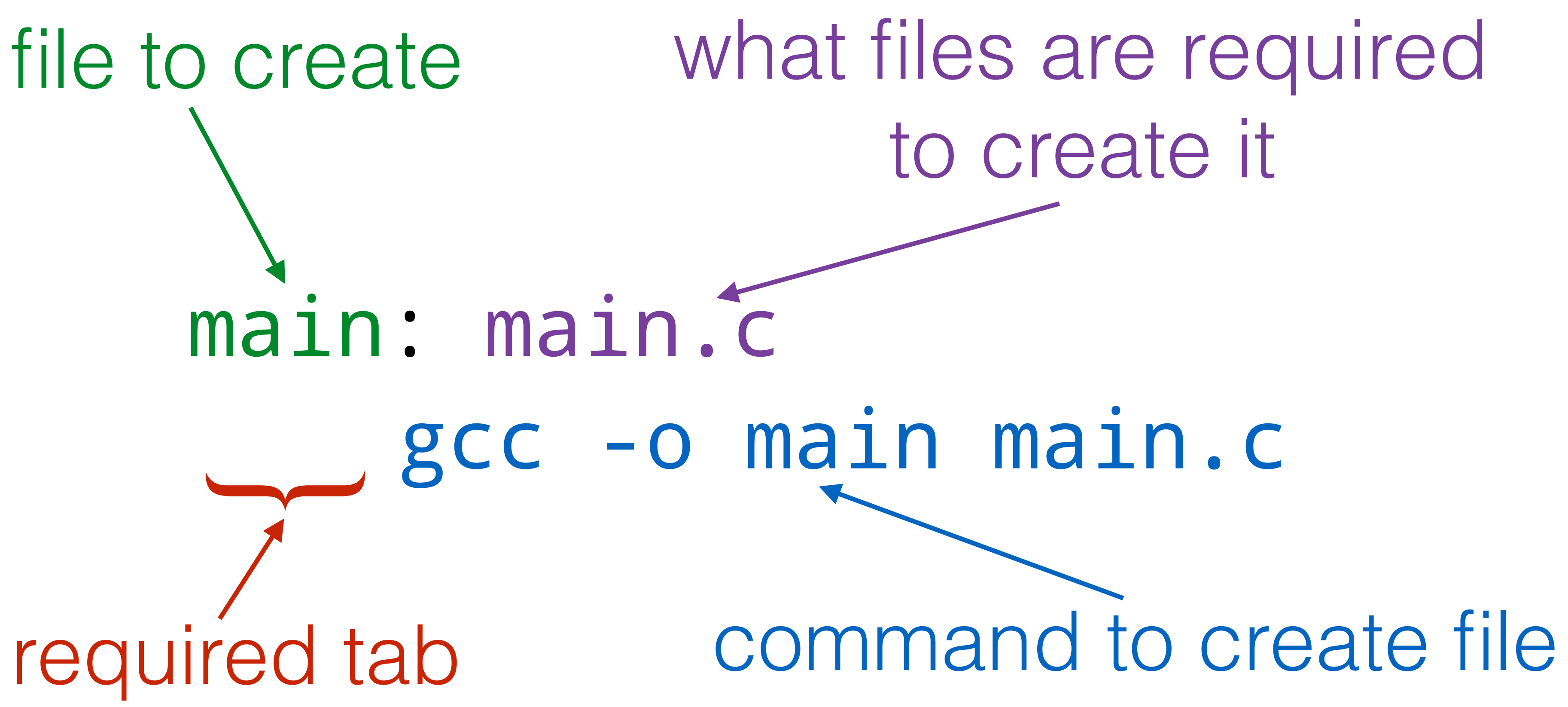
- You should know or be able to:
- Know what makefiles are for and how to write a basic makefile.
- Understand how they help with the incremental build process.
- Know how to create a rule for a specific target.
- Know how to run make from the command line, and how to run a specific target.
- Understand the automatic make variables: %, `$@, $^, $<, $?.`
- Know how to make a phoney target and why you may want to do that (c.f. clean, test).



```
# Comments in Make can look like this
CC=gcc
EXE=myprog

CFLAGS=-Wall
LDFLAGS=-lm

OPTCFLAGS=-O2
DEBUGCFLAGS=-g

OBJ=main.o util.o

# Default Target, dependency on $(EXE) target
all: $(EXE)

# Optimised target, add OPTCFLAGS
opt: CFLAGS+=$(OPTCFLAGS)
opt: $(EXE)

# Debug target, add DEBUG Flags
debug: CFLAGS+=$(DEBUGCFLAGS)
debug: $(EXE)

$(EXE): $(OBJ)
	$(CC) $(LDFLAGS) -o $@ $^

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

.PHONY: clean test
clean:
	rm *.o
	rm $(EXE)
```

```
Makefile                                                   1,1           All
"Makefile" 33L, 463C
```

Make Rules



```
main: main.c
	gcc -o main main.c
```

- Makefile rules require a **target**, it's **dependencies** and the **command** needed to produce the target from it's dependencies. In this instance `main` is produced from `main.c` by running `gcc`.

- Makefiles can be used for a number of things, not just compiling. For instance the following rule downloads the xkcd comic you saw before:

```
compiling.png:
	wget http://imgs.xkcd.com/comics/compiling.png
```

- The Filesystem
- Processes
- Bash Scripting
- Compilation
- Makefiles
- Git
- GDB

Git

- You should be able to:
- Understand the purpose of revision control.
- Understand the basic concepts used in revision control:
    - repository, checkout, commit, branch, merge
- Understand the difference between distributed and centralised revision control systems.
- Initialise a basic git repository:
    - `git init project`
    - `git clone https://www.bitbucket.org/p2t/myrepo`
- Add and remove files
    - `git add myfile.c`
    - `git rm myfile.c`
- The importance if the commit command and what it is used for.
    - `git commit -m “My first commit”`
- How to check the status and get the commit history:
    - `git status`
    - `git log --oneline`
- How to work with branches:
    - `git branch my feature`
    - `git branch --list`
    - `git checkout myfeature`
    - `git merge myfeature`

- The Filesystem
- Processes
- Bash Scripting
- Compilation
- Makefiles
- Git
- GDB

GDB

- You should be able to:
- Compile code such that it contains the required information to run gdb (gcc -g)
- Run gdb from the command line:
  - `gdb`
  - `gdbtui`
- Understand the basic gdb commands
  - `list, run, break, next, step, continue, print, watch`
- Have a basic understanding of how you could use gdb to debug a program.

- The Filesystem
- Processes
- Bash Scripting
- Compilation
- Makefiles
- Git
- GDB & debugging