

Introduction to Unix

COMP2017/COMP9017

FACULTY OF
ENGINEERING &
INFORMATION
TECHNOLOGIES



THE UNIVERSITY OF
SYDNEY



› a **kernel**

- program that handles the hardware directly (memory management, devices etc)
- provides an application program interface (API) for user and system programs
- provides a name space and file system

› a large set of utility programs

- system management
- command line interpreters (“shells”)
- file management
- text editors
- compilers

› documentation

- “manual entries”
-

The Command Interpreter or *shell*

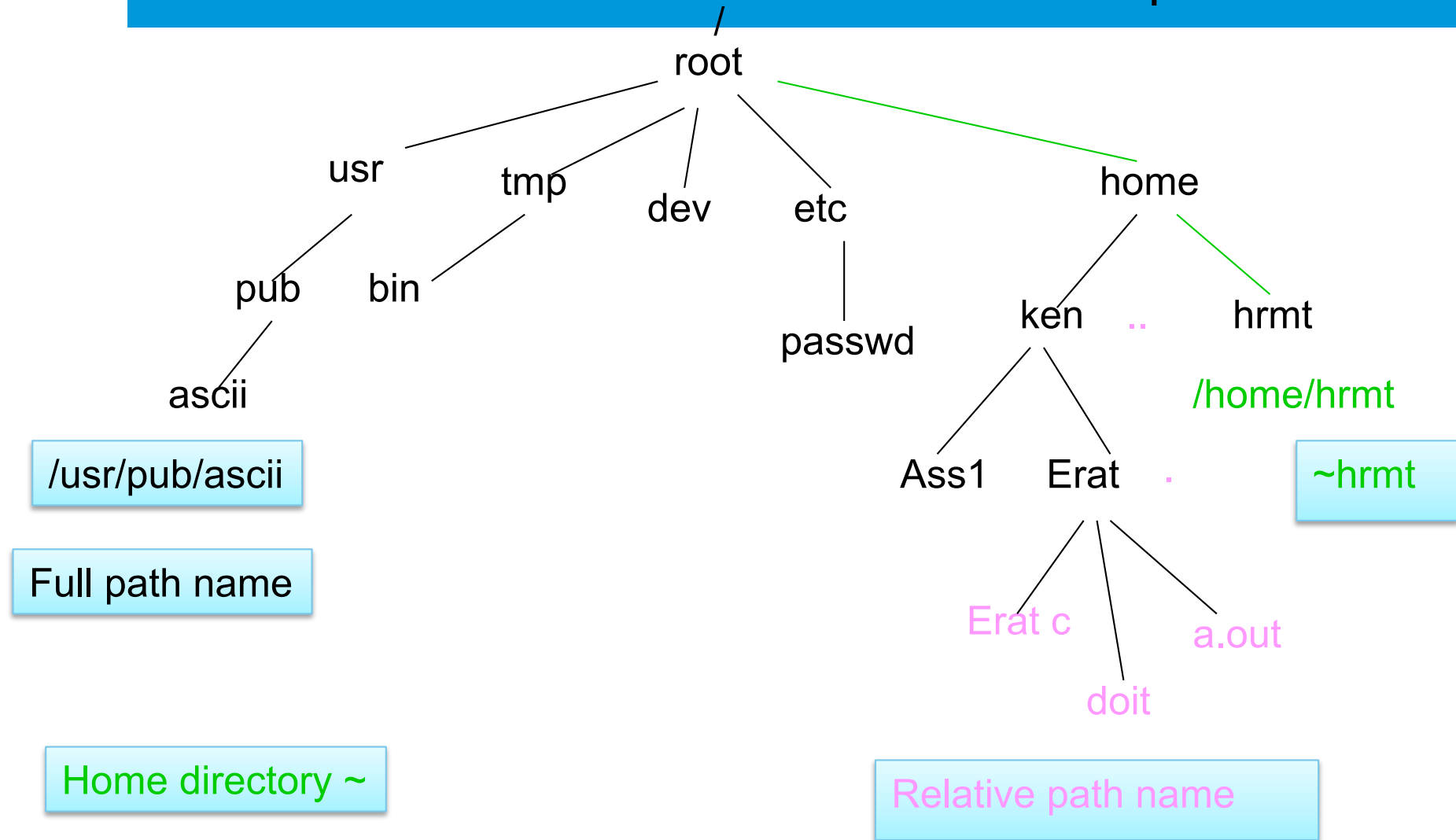
- › one of the first programs written for Unix was a program to interpret commands typed by a user
 - › there are many different command interpreters, but they all have the generic name of “shell”
 - › a shell reads a command, works out what program to run, and then runs the program with parameters specified
 - notice that the shell doesn't do anything except process parameters and run other programs
-

In the beginning was the **Command Line**

- › log in
 - › start a command window
 - that's all there was in the beginning....
 - › type text commands.....

 - › create a text file containing the C program
 - › compile it
 - › run it
 - › success!
-

Paths and full path names



› Several types of files

- Ordinary files
 - Contain data, eg text, program
- Directories
 - Actually a special file that contains a list of files
- Devices
 - The “file” is a name that refers to a device such as a disk drive or network interface
- “special” files
 - The content of these “files” is generated when the file is read
 - Eg the files in /proc on a linux system

```
bash$ ls -l
```

```
drwxr-x---    6 judy   judy    4096 Dec 14  2012 Cs12000
drwxrwxr-x   12 judy   judy    4096 Jul 10  2012 Cs12001
drwxrwxr-x    6 judy   judy    4096 Mar 15 12:29 EPS
drwxr-x---    4 judy   judy    4096 Feb 11  1999 IF
-rw-r-----    1 judy   judy    4882 Feb 12  2001 Kay.ps
drwxrwxr-x    9 judy   judy    4096 Mar 13 22:01 SDM
drwxr-x---    7 judy   judy    4096 Dec 17 11:06 Uidp
```

› Directory/file etc

› user, group, other

› Read

- File: program can read it
- Directory: can list directory contents (`ls`)

› Write

- File: program can alter it
- Directory: can add/remove files in it

› Execute

- File: can run the program
 - Directory: can read files in it, if name given
-

- › Your home directory

`ls -ld .`

- › Protecting files from yourself

`chmod u-w precious_file.c`

- › Making files executable

`chmod u+x doit`

General form of chmod symbolic commands

- › `chmod [u g o] [+ -] [r w x] filenames`
-

- › Your default shell and environment
 - *a shell reads a command, works out what program to run, and then runs the program with parameters specified*
 - › Paths
 - › “globbing”
 - › Software development with less effort, using shell scripts to compile and test code
-

› Shell reads from

`/etc/profile`

› Then it reads your own

`.bash_profile`

`.profile`

› Is your environment different?

› How can you see these files, starting with .

Initial shell on lab machines

```
bash$ env
```

```
LOGNAME=demo
```

```
LD_LIBRARY_PATH=/gnu/usr/lib:/usr/openwin/lib:/local/usr/lib:/usr/local/openssl/lib
```

```
WWW_HOME=http://www.ug.cs.usyd.edu.au/
```

```
TERM=vt100
```

```
HOSTTYPE=sparc
```

```
PATH=/local/usr/bin:/gnu/usr/bin:/usr/bin:/bin:/usr/ccs/bin:/usr/ucb:/usr/openwin/bin:./usr/bags/stubin
```

```
HOME=/usr/cs2/demo
```

```
SHELL=/gnu/usr/bin/bash
```

```
PS1=bash$
```

```
HZ=100
```

```
http_proxy=http://www-cache.cs.usyd.edu.au:8000/
```

```
ftp_proxy=http://www-cache.cs.usyd.edu.au:8000/
```

```
MANPATH=/local/usr/man:/gnu/usr/man:/usr/share/man:/usr/openwin/man
```

```
gopher_proxy=http://www-cache.cs.usyd.edu.au:8000/
```

```
OSTYPE=SunOS5
```

```
NNTPSERVER=news.cs.usyd.edu.au
```

```
OPENWINHOME=/usr/openwin
```

```
SHLVL=1
```

```
EDITOR=/local/usr/bin/red
```

```
TZ=Australia/NSW
```

```
WWW_http_GATEWAY=http://www-cache.cs.usyd.edu.au:8000/
```

```
_=/usr/bin/env
```

- › Default search paths for finding programs - \$PATH
- › echo \$PATH
 - colon separated list of paths
- › When you type `gcc` the shell will search all the directories in \$PATH until it finds the *first* file called `gcc`

PATH=

/local/usr/bin:

/gnu/usr/bin:

/usr/bin:

/bin:

/usr/ccs/bin:

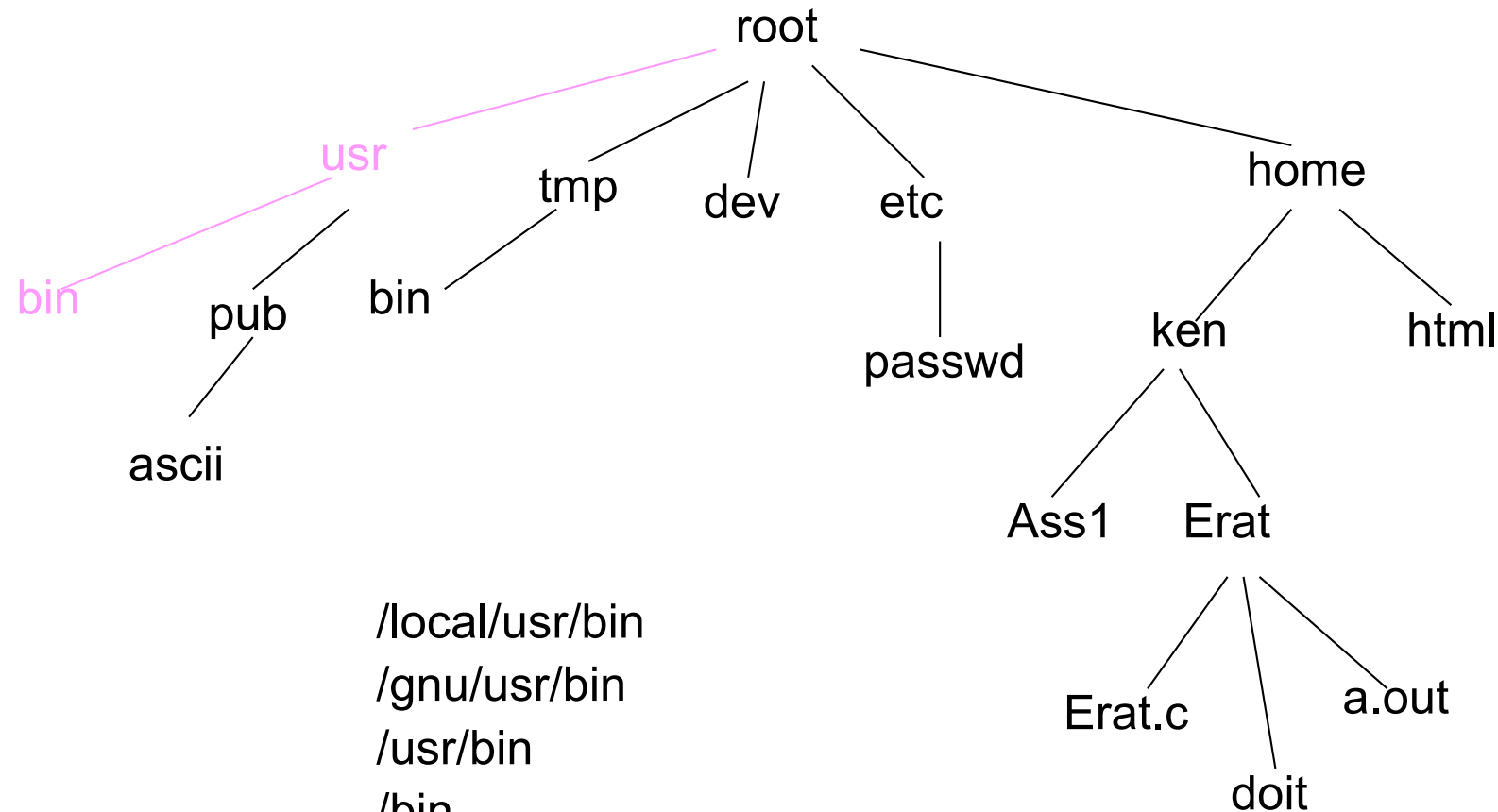
/usr/ucb:

/usr/openwin/bin:

:

..

Paths searched by shell



/local/usr/bin
/gnu/usr/bin
/usr/bin
/bin
/usr/ccs/bin
/usr/ucb
/usr/openwin/bin

```
bash$ gcc -ansi -wall -pedantic tests.c
```

```
bash$ gcc -ansi -wall -pedantic *.c
```

› general form in manual entry

```
gcc [ option | filename ]...
```

```
bash$ man gcc
```

```
bash$ which gcc
```

```
gcc -ansi -Wall -pedantic *.c
```

- › first word is the command name - `gcc`
 - › spaces are delimiters
 - › expansions of metacharacters *
 - › look for this along `$PATH` environment variable
 - › execute program with that name
 - › pass rest of line as parameters
-
- › programmer of command (`gcc`) decides what is allowed

- › Regular expressions for pattern matching

\$ man re_format

- › Selected shell short cuts

* [] 0-9 a-z

- › Examples

\$ ls *.c

\$ gcc *.c

\$ ls -ld [0-9]*

\$ ls [a-z]*.c

\$ rm -i *

› Man pages

- \$ `man gcc`
- \$ `man file`
- \$ `man ls`

› Online manual

› Practice with them to utilise from your memory

› `apropos` command is useful to find an available program e.g.

- \$ `apropos sort`
 - \$ `apropos pdf`
-

The Unix Shell: key concepts

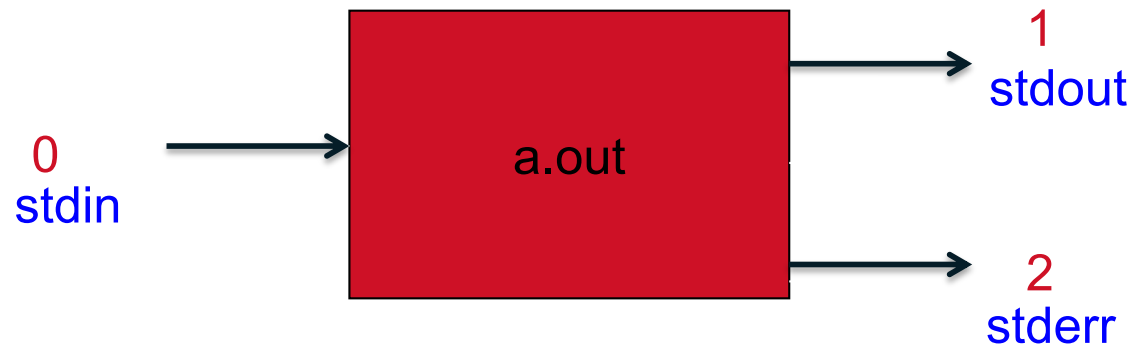
- › I/O redirection and pipes
 - › scripts
 - › variables
 - › control structures
-
- › these shell features are supported by many simple commands that can be combined in scripts to provide powerful services

- › when programs are run they have three standard files opened:
 - standard input (0)
 - standard output (1)
 - standard error (2)
 - › these are normally all connected to the terminal window
 - › can be **redirected** using > or < or |

```
gcc myprog.c > output
```
-

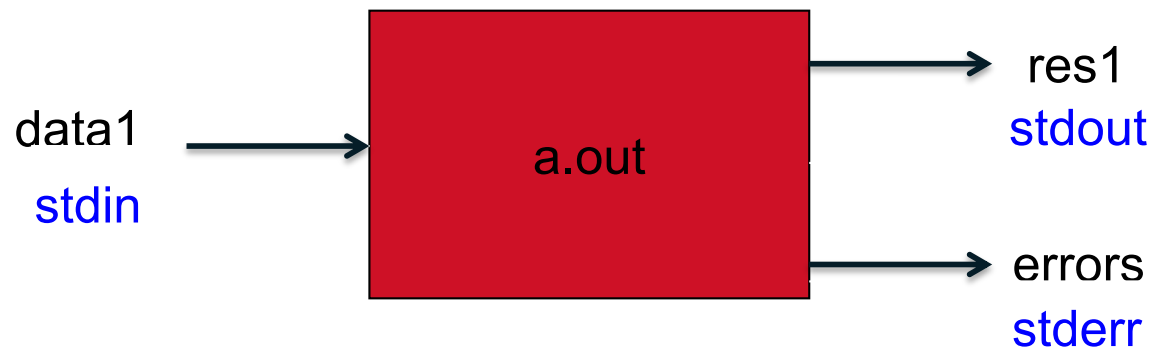


Redirection and pipes



bash\$./a.out

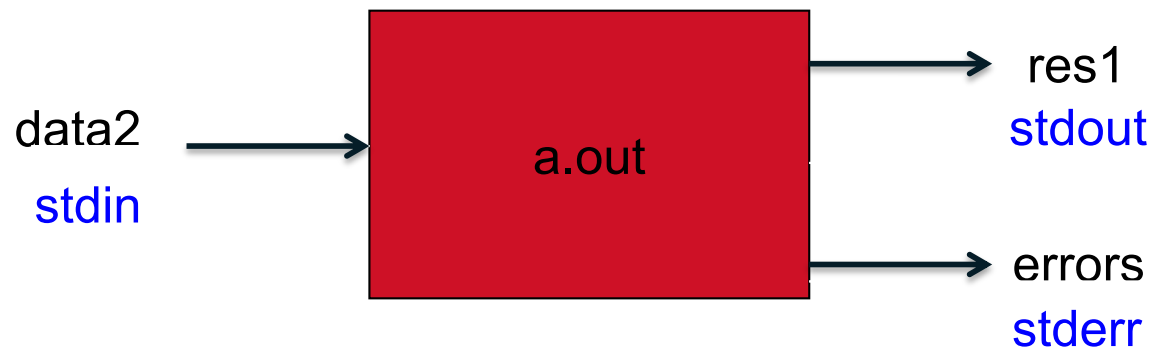
Fixed *file* numbers: 0, 1, 2



```
bash$ ./a.out
```

```
bash$ ./a.out < data1 > res1 2> errors
```

Redirection - appending



```
bash$ ./a.out
```

```
bash$ ./a.out < data2 >> res1 2>errors
```

Now, each run of the command does what?

- › pipes let you connect the standard output of one program to the standard input of another program

```
$ myprog | hisprog
```

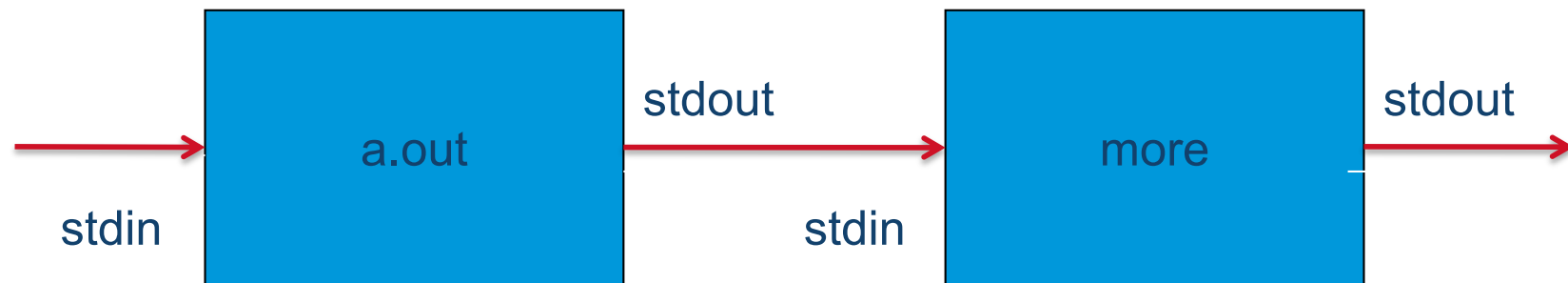
- › this has the same effect as

```
$ myprog > tempfile
```

```
$ hisprog <tempfile
```



Redirection and pipes



a.out | more

“pipe” symbol

- › Create a command that will show all the details of the 5 most recently changed files in the current directory

```
$ ls -ltr | tail -5
```

- › Need execute permission...
 - › Need it on path...
-

- › commands can be placed in a file and the shell will read and interpret them
- › this allows you to create new commands easily:

```
#!/bin/sh  
  
gcc myprog.c -o myprog  
echo Finished!
```

which command
interpreter to use

- › script file should be readable and executable

```
gcc -W -Wall -pedantic -ansi erat.c
a.out < 01_data > 01_res
a.out < 02_data > 02_res
a.out < 03_data > 03_res
echo "comparing first test results ....."
diff 01_res 01_expected
echo "comparing second test results ....."
diff 02_res 02_expected
echo "comparing third test results ....."
diff 03_res 03_expected
```

- › Then make doit executable
 - › If doit not in \$PATH, need ./doit, else doit
 - › What about 50 tests?
-

```
gcc -W -Wall -pedantic -ansi erat.c  
echo "Test on 01_data - normal" > result  
a.out < 01_data >> result  
printf "Test on 02_data - normal\n" >> result  
a.out < 02_data >> result  
echo "Test on 03_data - boundary" >> result  
a.out < 03_data >> result
```

- › Saves typing
- › Means you will be more likely to test systematically
- › ... you know there has to be a better way to handle this, especially with 50..100.. tests

- › the shell has *variables* that you can give a string value to
 - › there are a number of predefined variables such as PATH and HOME
 - › you can create your own variables in a script or on the command line
 - › the variables are stored in the *environment* of a program, on the stack along with program arguments when the program starts
-

- › you set a shell variable with a statement like:

`VARNAME=value`

- › you use a shell variable in a statement like:

`$VARNAME`

- › the shell will replace the `$VARNAME` with the string value
-

- › shell script arguments are available in special variables \$1, \$2 etc

- › for example, the script called “compile”:

```
#!/bin/sh
```

```
gcc $1.c -o $1
```

- › you invoke this like:

```
compile myprog
```

- › a list of all script arguments is available in the shell variable \$*

- › there are many different shell command interpreters available
 - › Most shells offer the control structures:
 - if
 - for
 - case
 - while
 - › these are implemented by the shell program itself, not by separate commands
-

Useful Commands for handling text files

<code>sort</code>	sorts lines of text in a file, very flexible choice of key field to sort on, can remove duplicates, sends sorted lines to output
-------------------	----------------------------------------------------------------------------------------------------------------------------------

<code>cut</code>	cuts fields out lines of text from a file and sends the result to output
------------------	--------------------------------------------------------------------------

<code>tr</code>	transliterates or removes characters from a file eg remove <code>\r</code> from a file
-----------------	-------------------------------------------------------------------------------------------

<code>comm</code>	compares files and prints lines that appear in only one or both the files
-------------------	---------------------------------------------------------------------------

see the manual entries for details

- › shell command language has I/O redirection, pipes, control structures and variables
 - › there are many shell commands that do useful operations on files of text
 - › shell scripts can be written using commands and the command language
 - › picture acknowledgement:
<http://royshort.com/DukeP.htm>
-