Shell basics

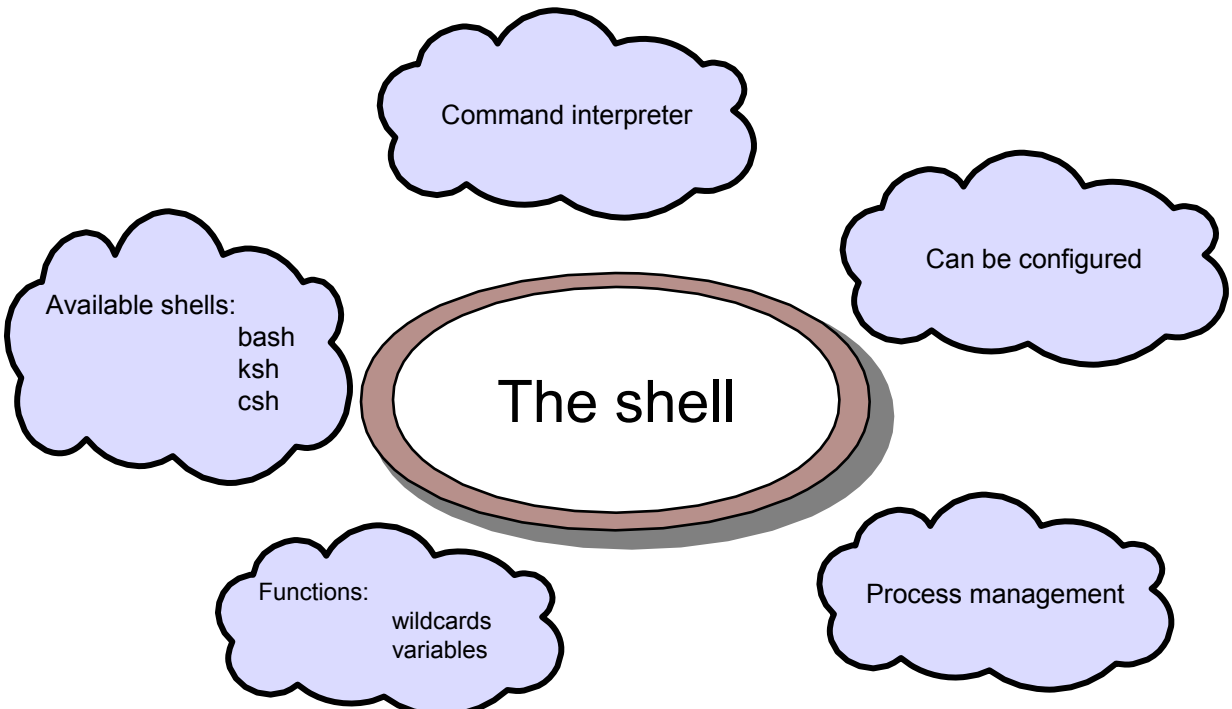
Unit objectives

After completing this unit, you should be able to:

- Explain the function of the shell
- Discuss metacharacters and reserved words
- Use wildcards to access files with similar names
- Use redirection and pipes
- Use command substitution
- Describe and use the most common filters
- Group commands to control their execution
- Work with shell variables
- Use aliases
- Apply quoting

The shell

- The *shell* is the user interface to Linux.



Shell features

- When the user types a command, various things are done by the shell before the command is actually executed.
  - Wildcard expansion — * ? [ ]
  - Input/output redirection — < > >> 2>
  - Command grouping — { com1 ; com2; }
  - Line continuation — \
  - Shell variable expansion — $var
  - Alias expansion — dir -> ls -l
  - Shell scripting — #!/bin/bash

- For example, the `ls *.doc` command could be expanded to `/bin/ls --color=tty mydoc.doc user.doc` before execution (depending on settings and files present).

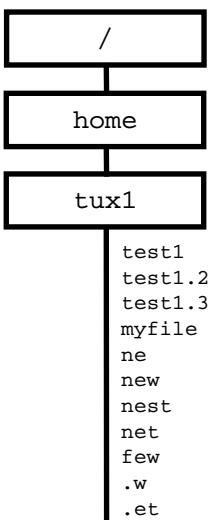
Metacharacters and reserved words

- Metacharacters are characters that the shell interprets as having a special meaning.
  - Examples: < > | ; ! ? * $ \ ` ' " ~ [] ( ) {}
- Reserved words are words that the shell interprets as special commands.
  - Examples: case, do, done, elif, else, esac, for, fi, function, if, in, select, then, until, while

Basic wildcard expansion

- When the shell encounters a word that contains a wildcard, it tries to expand this to all matching file names in the given directory.

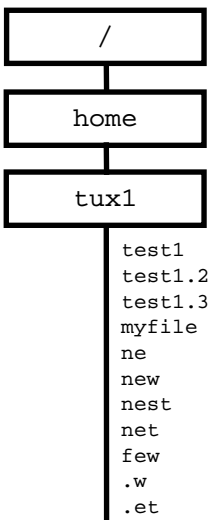
```
$ ls -a
. .. .et .w few myfile ne nest net new test1 test1.2
test1.3

? matches a single character
$ ls ne?
net  new
$ ls ?e?
few  net  new

* matches any string, including the null string
$ ls n*
ne  nest  net  new
$ ls *w
few  new
```



Advanced wildcard expansion

- The wildcards [,], -, and ! match inclusive lists.

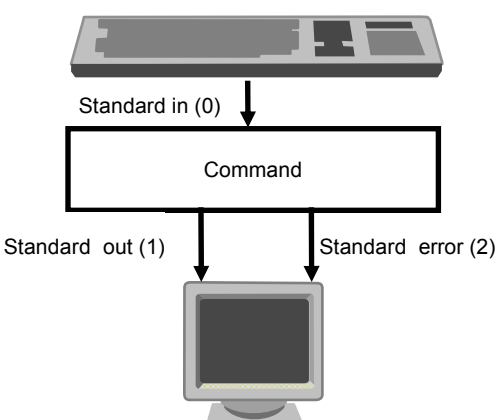
```
$ ls ne[stw]
net  new

$ ls *[1-5]
test1  test1.2  test1.3

$ ls [!tn]*
few  myfile

$ ls ?[!y]*[2-5]
test1.2  test1.3
```



File descriptors

- Every program has a number of file descriptors associated with it.
- Three descriptors are assigned by the shell when the program starts (STDIN, STDOUT, and STDERR).
- Other descriptors are assigned by the program when it opens files.

Standard in	STDIN	<	0
Standard out	STDOUT	>	1
Standard error	STDERR	2>	2



Input redirection

- Default standard input:

```
$ cat
Atlanta
Atlanta
Chicago
Chicago
<Ctrl-d>
```

- STDIN redirected from file:

```
$ cat < cities
Atlanta
Chicago
$
```

Output redirection

- Default standard output: `/dev/tty`

```
$ ls
file1 file2 file3
```

- Redirect output to a file:

```
$ ls > ls.out
```

- Redirect and append output to a file:

```
$ ls >> ls.out
```

- Create a file with redirection:

```
$ cat > new_file
Save this line
<Ctrl-d>
```

Error redirection

- Default standard error: `/dev/tty`

```
$ cat fileA
cat: fileA: No such file or directory
```

- Redirect error output to a file:

```
$ cat fileA 2> error.file
$ cat error.file
cat: fileA: No such file or directory
```

- Redirect and append errors to a file:

```
$ cat fileA 2>> error.file
```

- Discard error output:

```
$ cat fileA 2> /dev/null
```

Combined redirection

- Combined redirects

```
$ cat < cities > cities.copy 2> error.file
$ cat >> cities.copy 2>> error.file < morecities
```

- Association
  - This redirects STDERR to where STDOUT is redirected.

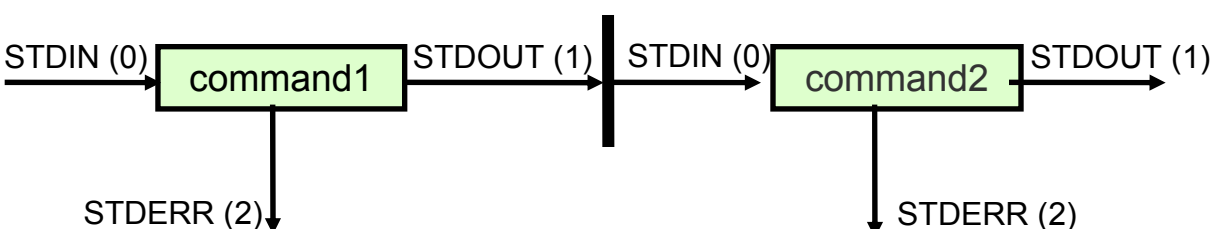
```
$ cat cities > cities.copy 2>&1
...writes both stderr and stdout to cities.copy

be careful about this:
$ cat cities 2>&1 > cities.copy
...writes stderr to /dev/tty and stdout to cities.copy
```

Pipes

- A sequence of two or more commands separated by a vertical bar (|) is called a *pipe* or *pipeline*.

```
$ ls -l | wc -l
```

- The standard output of command1 becomes the standard input of command2.



Filters

- A *filter* is a command that reads from standard in, transforms the input in some way, and writes to standard out. They can, therefore, be used at intermediate points in a pipeline.

```
$ ls | grep .doc | wc -l
4
```

Common filters

- **grep:** Only displays lines that match a pattern
- **sed:** Allows string substitutions
- **awk:** Pattern scanning and processing
- **fmt:** Insert line wraps so that text looks pretty
- **expand**, **unexpand:** Change tabs to spaces and vice versa
- **tr:** Substitute characters
- **nl:** Number lines
- **pr:** Format for printer
- **sort:** Sort the lines in the file
- **tac:** Display lines in reverse order

Split output

- The **tee** command reads standard input and sends the data to both standard output and a file.

```
$ ls | wc -l
3

$ ls | tee ls.save | wc -l
3

$ cat ls.save
file1
file2
file3
```

Command substitution

- Command substitution allows you to use the output of a command as arguments for another command.
- Use backticks -- ` -- or $() notation.

```
$ rm -i `ls *.doc | grep tmp`

$ echo There are $(ps ax | wc -l) processes running.
```

Command grouping

- Multiple commands can be entered on the same line, separated by a semicolon (;).

```
$ date ; pwd
```

- Commands can be grouped into one input/output stream by putting curly braces ({ }) around them.

```
$  { echo Print date: ; date ; cat cities; }  | lpr
```

- Commands can be executed in a subshell by putting round braces () around them.

```
$ ( echo Print date: ; date ; cat cities ) | lpr
```

Shell variables

- Variables are part of the shell you are running.
- A variable has a unique name.
- The first character must not be a digit.
- To assign a value to a variable use `variable=value`.

```
$ var1="Hello class"

$ var2=2
```

Referencing shell variables

- To reference the value of a variable, use `$variable`.

```
$ echo $var1
Hello class

$ echo $var2
2
```

Exporting shell variables

- The **export** command is used to pass variables from a parent to a child process by putting it in the environment of the child process.
- Changes made to variables in a child process do not affect the variables in its parent.

```
$ export x=4
$ bash
$ echo $x
4
$ x=100
$ echo $x
100
$ exit
$ echo $x
4
```

Standard shell variables

- The shell uses several shell variables internally.
- These variables are always written in uppercase.
- Examples include the following:
  - **$:** PID of current shell
  - **PATH:** Path which is searched for executables
  - **PS1:** Primary shell prompt
  - **PS2:** Secondary shell prompt
  - **PWD:** Current working directory
  - **HOME:** Home directory of user
  - **LANG:** Language of user
- Overwriting these and other system variables by accident can cause unexpected results.
- Use lowercase variables in your shell scripts to avoid conflicts.

Return codes from commands

- A command returns a value to the parent process. By convention, zero means success and a non-zero value means an error occurred.
- A pipeline returns a single value to its parent.
- The environment variable question mark (?) contains the return code of the previous command.

```
$ whoami
tux1
$ echo $?
0
$ cat fileA
cat: fileA: No such file or directory
$ echo $?
1
```

Quoting metacharacters

- When you want a metacharacter *not* to be interpreted by the shell, you need to quote it.
- Quoting a single character is done with the backslash (\).

```
$ echo The amount is US\$5
The amount is US$5
```

- Quoting a string is done with single (') or double (") quotes.
  - Double quotes allow interpretation of the dollar sign ($), backtick (`), and backslash (\).

```
$ amount=5
$ echo 'The amount is $amount'
The amount is $amount
$ echo "The amount is $amount"
The amount is 5
```

Quoting non-metacharacters

- The backslash can also be used to give a special meaning to a non-metacharacter (typically used in regular expressions).
  - **\n:** New line
  - **\t:** Tab
  - **\b:** Bell
- A backslash followed directly by **Enter** is used for line continuation.
  - The continued line is identified with the $PS2 prompt (default: >).

```
$ cat/home/tux1/mydir/myprogs/data/information/letter\
> /pictures/logo.jpg
```

Aliases

- The **alias** command allows you to set up aliases for often-used commands.
- For example:

```
$ alias ll='ls -l'
$ alias rm='rm -i'
To show all currently defined aliases:
$ alias
To delete an alias:
$ unalias ll
$ ll
bash: ll: command not found
```

Unit review

- The shell is the command interpreter of Linux.
- The default shell in Linux is bash.
- A shell has a number of additional features, such as wildcard expansion, alias expansion, redirection, command grouping, and variable expansion.
- Metacharacters are a number of characters that have a special meaning to the shell.
- Reserved words are words that have a special meaning to the shell.

Checkpoint

1. True or False: A filter is a command that reads a file, performs operations on this file and then writes the result back to this file.

2. The output of the **ls** command is:

```
one   two   three   four   five
six   seven   eight   nine   ten
```

What will the output be if you run the `ls ?e*` command?

a. `three seven ten`
b. `seven ten`
c. `one three five seven eight nine ten`
d. `?e*`

Checkpoint solutions

1. True or False: A filter is a command that reads a file, performs operations on this file and then writes the result back to this file.

   The answer is false.

2. The output of the **ls** command is:

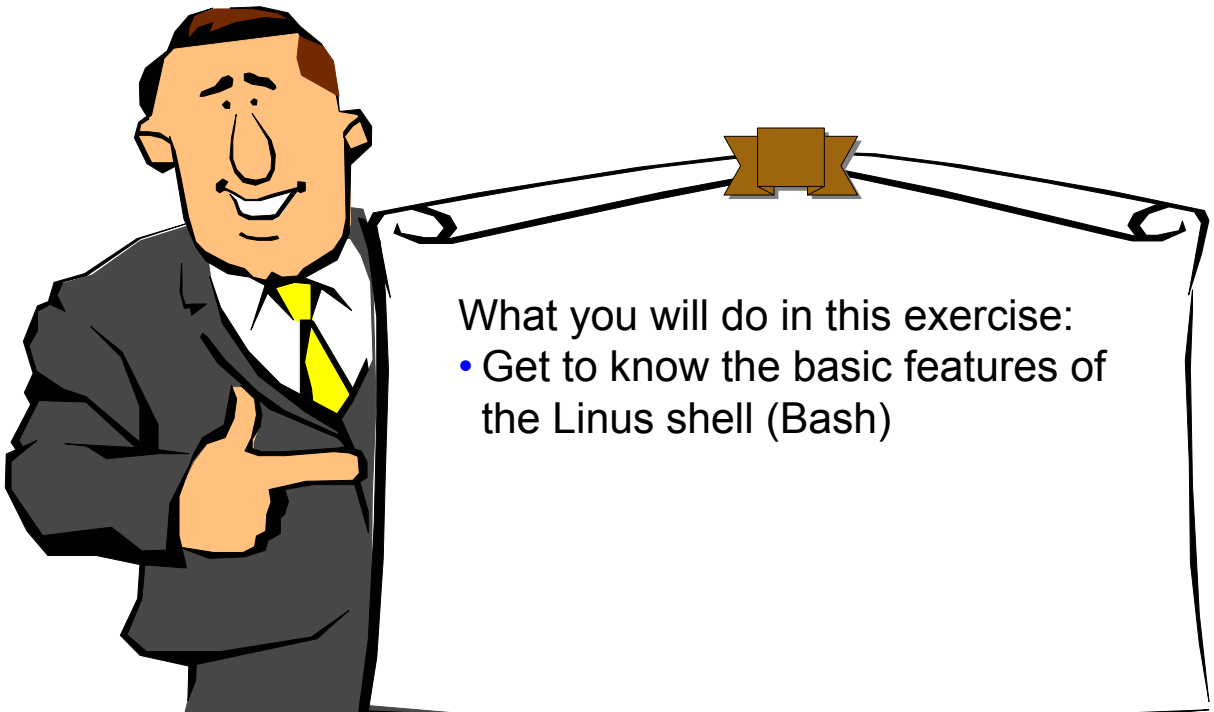
   ```
   one   two   three   four   five
   six   seven   eight   nine   ten
   ```

   What will the output be if you run the `ls ?e*` command?

   a. `three seven ten`
   b. `seven ten`
   c. `one three five seven eight nine ten`
   d. `?e*`

   The answer is `seven ten`.

Exercise: Shell basics



Unit summary

Having completed this unit, you should be able to:

- Explain the function of the shell
- Discuss metacharacters and reserved words
- Use wildcards to access files with similar names
- Use redirection and pipes
- Use command substitution
- Describe and use the most common filters
- Group commands to control their execution
- Work with shell variables
- Use aliases
- Apply quoting