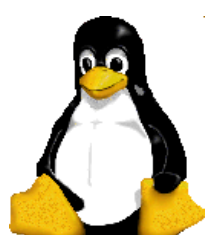
Shell Basics



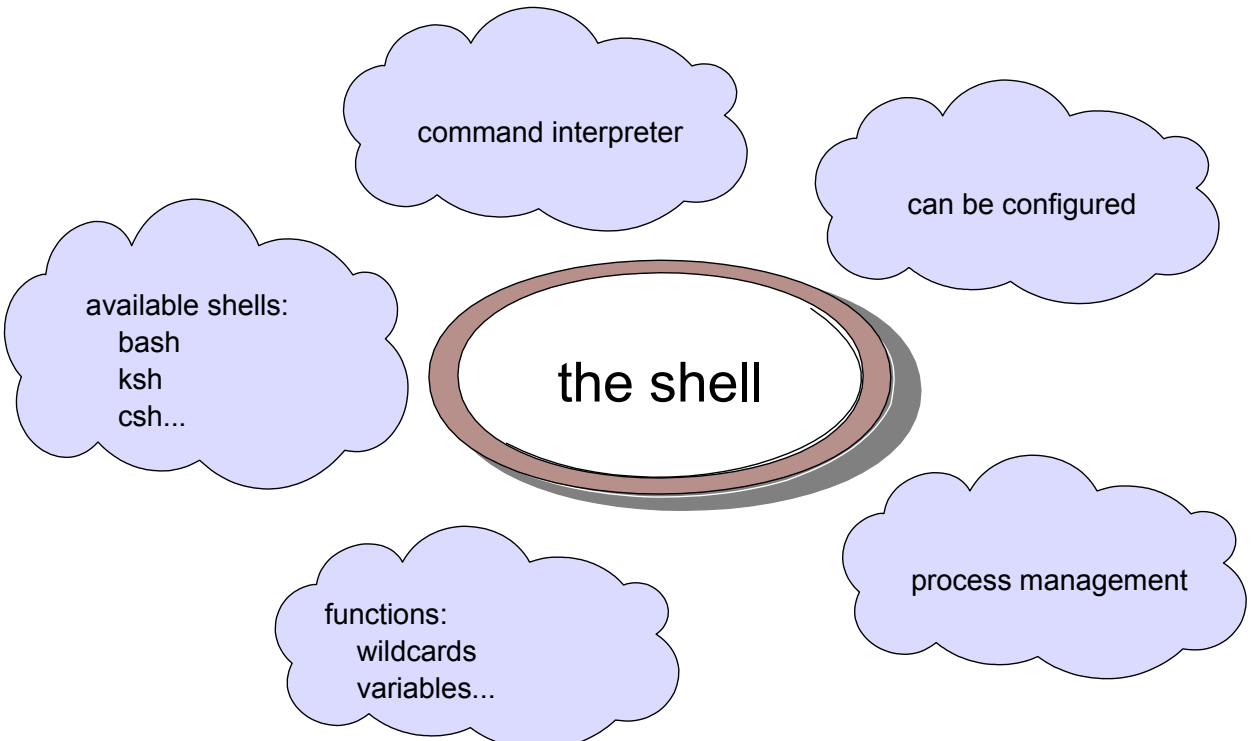
Objectives

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 in order to control their execution
- Work with shell variables
- Apply quoting
- Use aliases

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:

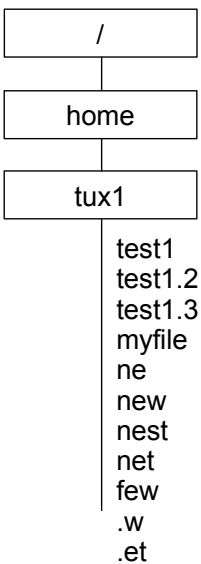
case do done elif else esac fi for function

if in select then until while

Basic Wildcard Expansion

- When the shell encounters a word which contains a wildcard, it tries to expand this to all matching filenames in the given directory

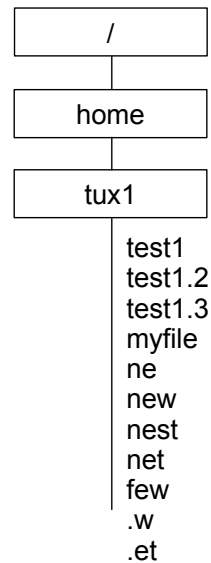
```
$ ls -a /home/tux1
. .. .et .w few myfile ne nest net new test1 test1.2 test1.3

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

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



Advanced Wildcard Expansion

- The wildcards **[**, **]**, **-** and **!** match inclusive lists:

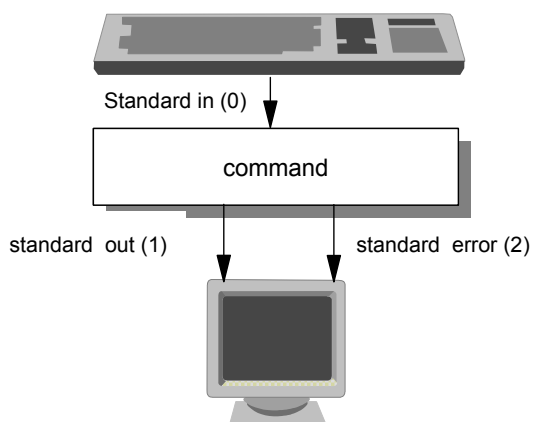
```
$ echo ne[stw]
net  new

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

$ echo [!tn]*
myfile few

$ echo ?[!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
Amsterdam
Amsterdam
Utrecht
Utrecht
<ctrl-d>
```

- STDIN redirected from file:

```
$ cat < cities
Amsterdam
Utrecht
$
```

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:

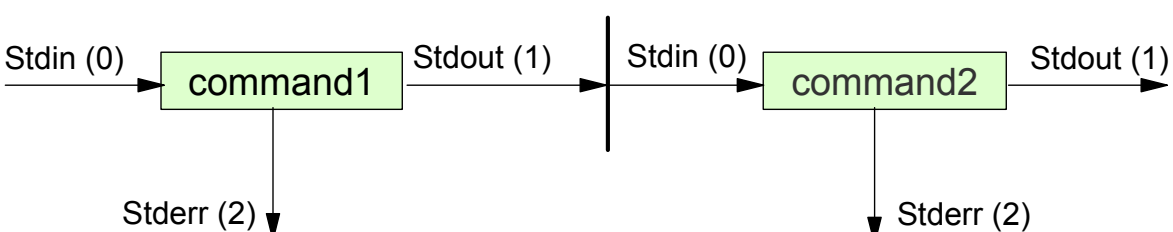
```
$ cat cities > cities.copy 2>&1
...writes both stderr and stdout to cities.copy

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

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

Split Output

- The **tee** command reads standard input and sends the data to both standard out 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 environment of a process
- A variable has an 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.
- 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
- Example:
  - **$**: 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 variables by accident can cause unexpected results
  - Always use lowercase variables in your own 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 **?** 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 $, `(backtick) and \

```
$ 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 = newline
  - \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/john/mydir/mysudir/data/information/letter\
> /pictures/logo.jpg
```

Aliases

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

```
$ 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 Summary

- 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, 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