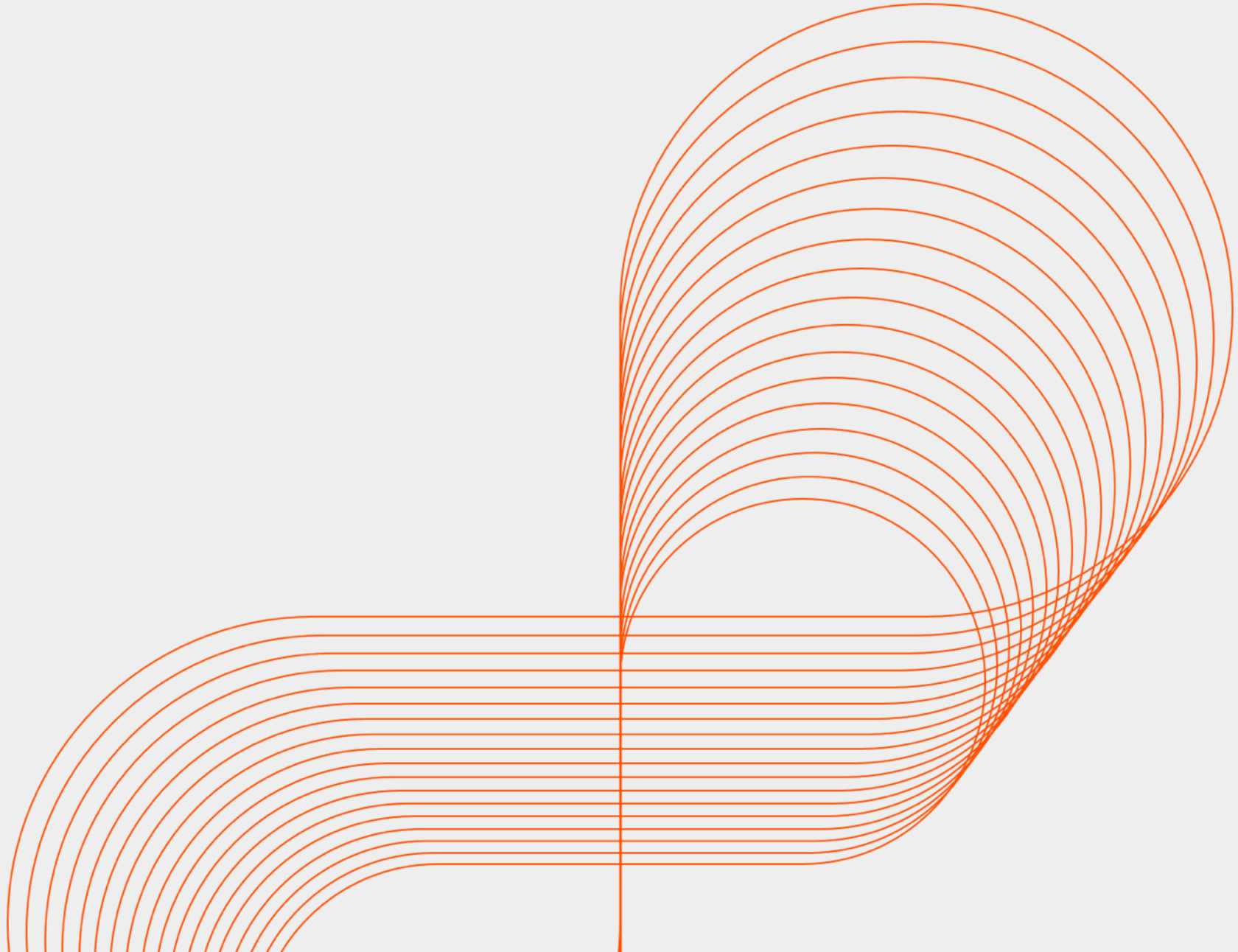
Persistent

UNIX



Command Modification

- Metacharacters
- Command Redirection
- Pipes

Metacharacters

- Command-line interface has a capability to use special characters called metacharacters to alter a command’s behavior
- There characters are not a part of the commands themselves, but are features of the shell that enable the user to create complex behaviors
- The most common metacharacters are wildcards
- These are special characters that can be used to match multiple files at the same time, increasing the chance that a command will find the desired file on the first try

Metacharacters

- The three wildcards that are used more often are
  - ? - matches any one character in a filename
  - * - matches any character or characters in a filename
  - [ ] - matches one of the characters inside the [ ] symbols
- These wildcards can be used in combination with a command to locate multiple files, or to find a file when you can’t quite remember its full name

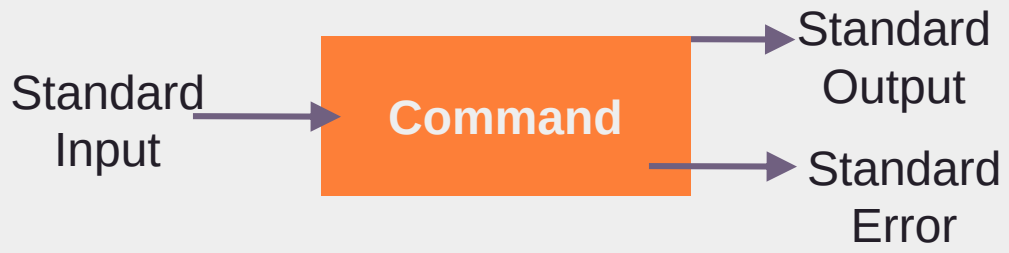
Examples

- Assume your directory contains the following files - *user1, user2, user3, mail.doc, info.doc, getpwd.c, streams.cpp, streams.o, myprog.f, myprog.o,, myprog.txt, data.text*
- The following table shows usage of some wildcards

Argument+Wildcard	Files Matched
user?	user1 user2 user3
*.doc	mail.doc info.doc
*	all files
streams*	streams.cpp streams.o
myprog[fo]	myprog.f myprog.o
.t[ex]	myprog.txt data.text

Standard I/O

- Each program that is invoked has three standard I/O channels
  - Standard input (also called stdin, usually the keyboard)
  - Standard output (also called stdout, usually the terminal)
  - Standard error (also called stderr, usually the terminal)



Standard I/O

- These streams are referred to by numbers called as file descriptors
- A file descriptor is a number that the OS assigns to a file to keep track of it
- The following file descriptors are assigned to these streams:
  - stdin - 0
  - stdout - 1
  - stderr - 2
- Numbers greater than 2 are assigned to user & system files

Command Redirection

- Every command needs a source of input and a destination for the output
- These attributes are programmed into the commands as default behavior
- Mostly, the standard input is the keyboard and the standard output is the screen
- In certain cases, you might want a command to take input from a file and dump the output into another file. This is called as redirection and is a great way of streamlining a sequence of tasks

I/O Redirection

- I/O redirection uses the following characters to define the temporary input and output source
  - > - redirect stdout from a command to a file on disk
  - >> - append output from a command to a file on disk
  - < - take stdin from a disk file
  - | - pass the output of one command to another for further processing
  - tee - used in the middle of a pipeline, this command allows you to both redirect output to a file and pass it to further commands in the pipeline
  - 2> - stderr is redirected to a file
  - 1> - stdout is redirected to a file
  - &> - stdout & stderr are redirected to the same file

Examples

- ps -e > processes.txt
- wc -l < processes.txt
- sort < unsorted > sorted
- cat users >> finallist
- ls *.txt *.log 2> errors
- ls *.cpp *.h &> sourcefiles

/dev/null

- By default, the errors generated while executing a command/commands are shown on the screen.
- The errors might interrupt your work by showing up on the screen at any time
- You may want to ignore the errors in such cases
- Simply redirecting them to a file is not the best solution, rather redirect them to */dev/null*
- It is also called null device and is a special file that discards all the data written to it, but reports the write operation has succeeded
- It is useful for disposing of unwanted output streams of a process, or as a convenient input file for input streams

Pipe

- Assume you want to sort the names of all the people that are currently logged in and display the output on the screen
- The command *who* gives the names of the logged in users and sort command will sort all the names
- The following commands will sort all the currently logged in users

  who > users

  sort users
- This requires a temporary file *users*
- Through pipes it can easily be achieved as follows:

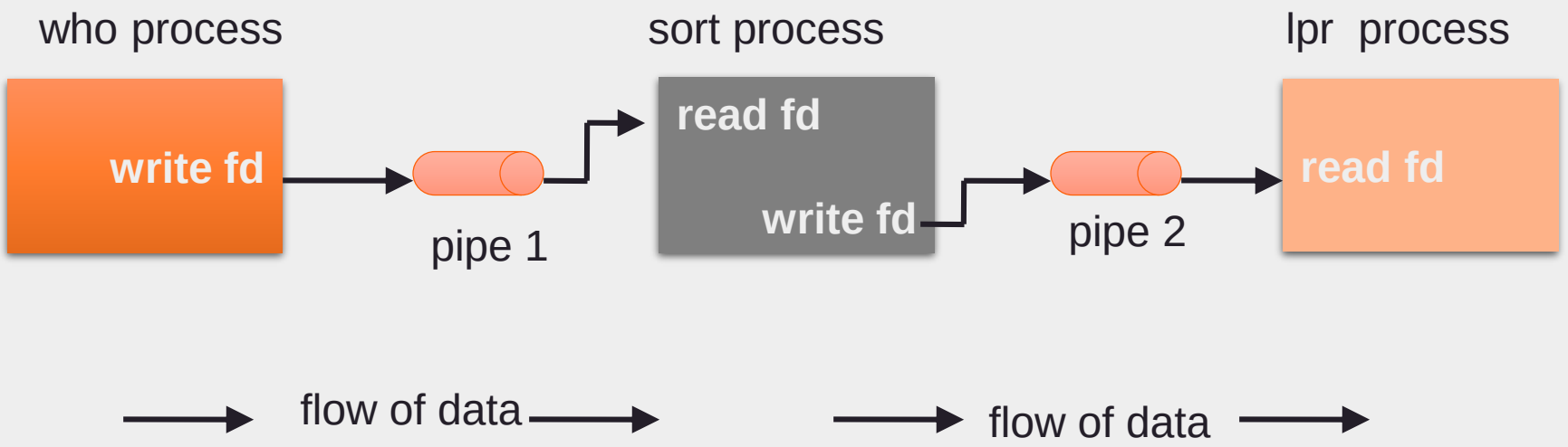
  who | sort

Pipe

- A pipe is a method of IPC which allows a one-way flow of information between two process on the same machine
- Many Unix commands take text input and/or produce text output. It’s sometimes useful to be able to control where the input comes from and output goes.
- This can be used to pass the output of one command as an input to other command
- A pipe is represent with the | symbol

Pipe

- If a Unix user issues the command *who*| *sort* | *lpr*, then the Unix shell would create three processes with two pipes between them



Persistent

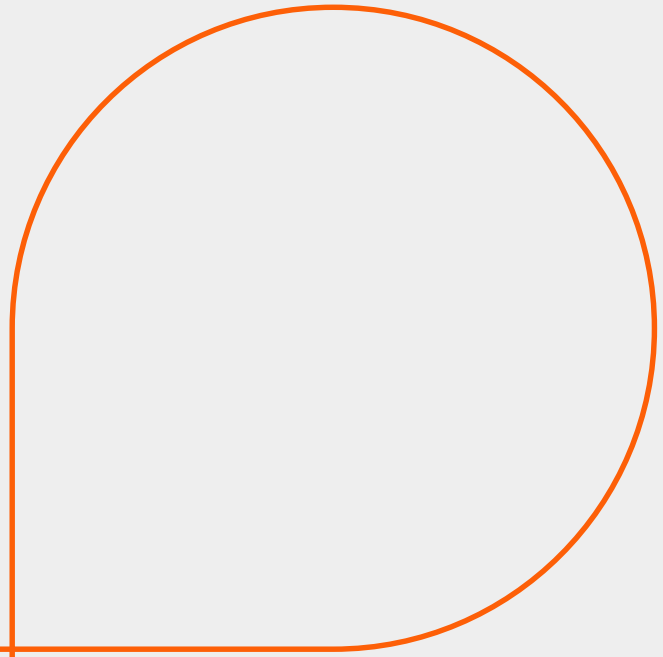
Examples

- ll *.cpp | tee output | wc –l
- grep /bin/bash /etc/passwd | wc -l

Assignments

- Write the names of all the users belonging to a particular group
- Create a file with names of all the users using bash shell
- Display the names of logged in users

Shell Scripting

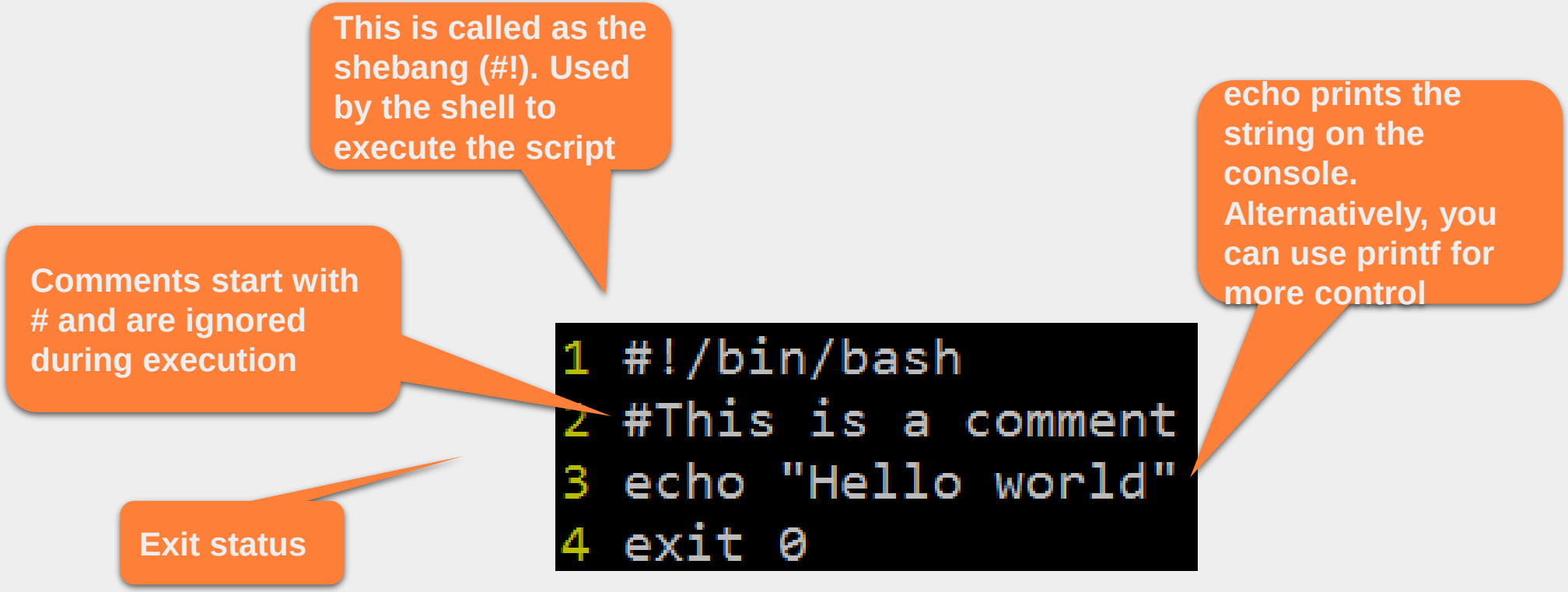


Shell Script

- A shell script is a text file that contains a sequence of commands
- A shell script is run by the shell for which it is written
- Shell scripts created for one shell might not run on a different shell due to incompatibilities

```
#!/bin/bash
#This script creates multiple users with random passwords
#This information is written to a text file called del.txt
#Umar Majid 10/4/2009
echo -e UserName "\t" Password >> del.txt
for var in $(seq 1 5)
        do
        number=$RANDOM
        echo -e $var "\t\t" $number >> del.txt
        done
exit 0
```

Hello World



This script can be run from the command line as :
#bash hello.sh
or if you make it executable:
#./hello.sh

Scripting Requirements

- You should follow these guidelines while writing scripts
  - Give it a unique name
  - Include the shebang
  - Include lots of comments
  - Use the exit command to indicate exit status
  - Make your scripts executable

Variables

- Lot of variables are automatically defined when shell is started e.g. name of computer or shell, history file, etc
- These are called as shell variables and are useful for referring to some value, or getting the data from the user or simply a value that is calculated dynamically
- With scripts each variable has a name & a value. The value of the variable is accessed using $
- Some built-in variables are 1-9 and refer to the command line arguments

Variables Example

```
#!/bin/bash
DIR=/root

ls $DIR
cd $DIR
pwd
exit 0
```

Taking Input

```
#!/bin/bash
#Take input from the user
echo "Enter you name"
read NAME
echo -e "Your name is:\t $NAME"
exit 0
```

Control Structures

- Shell supports the following control structures
  - if - used to execute commands only if certain conditions are met
  - case - used to work with options
  - for - used to run a command for a given number of items
  - while - use while as long as the specified condition is met
  - until - opposite of while. Used to run a command until a certain condition has been met

Comparison Operators

- Following operators can be used to compare expressions through the *test* or *[* command
- If the expression is true, it returns 0, otherwise 1

Comparison	Numerical	String
equal	-eq	=
Not equal	-ne	!=
Less than	-lt	<
Greater than	-gt	>
Less than or equal to	-le	
Greater than or equal to	-ge	

Shell Arithmetic

- Bash supports basic mathematical operations through the expr command
- The following operators are supported - +, -, /, * & %
- Another way of performing calculations is by using the shell built-in syntax $(( ))

e.g. x=`expr 3 + 10` [through command substitution]

x=$((3+10)) [built-in shell syntax]

Assignment

- Create a shell script to add two numbers taken from the user
- Modify the previous example to take the numbers from the command line

Links for objective multiple choice questions.

- http://www.sanfoundry.com/linux-command-mcq-1/
- http://www.sanfoundry.com/linux-command-mcq-2/
- http://www.sanfoundry.com/linux-command-mcq-3/
- http://www.indiabix.com/computer-science/unix/
- http://www.avatto.com/computer-science/test/mcqs/questions-answers/unix/153/1.html
- http://www.gkseries.com/computer-engineering/unix/multiple-choice-questions-and-answers-on-unix-and-shell-programming
- http://www.withoutbook.com/online_test.php?quiz=38&quesNo=10&subject=Top%2010%20UNIX%20Online%20Practice%20Test%20%7C%20Multiple%20Choice