Shell

Matt Warner

1 UNIX Command Interpreters

A command Interpreter is commonly refered to as a **shell**
Ever UNIX system has a "Bourne shell compatible" shell
The shell is where is simply where your commands are typed, and then interpreted.

1.1 History

- sh: original Bourne shell, written in 1978 by Steve Bourne
- ash: Almquist shell, BSD-licensed replacement of sh

1.2 Today

- bash: Bourne-again shell, GNU replacement of sh
- dash: Debian Almquist shell, small scripting shell

2 Variables

The shell remembers values stored in variables.
Variables have a name and a type, these types are limited to: **strings**, **numbers**, **arrays**.

To set a string variable we would write

```
$ variable=value
```

To display our variable we would write,

```
$ echo $variable
```

We can also change the value stored in our variable by just repeating the same syntax used to create it.

2.1 Variable Scope

Varible holds values for the duration of shell invocation, i.e., when the shell is exited, the variable goes away.

Variables can be exported into enviroment:
The enviroment is basically a set of key,value pairs, where the key is the variable name and the value if the value stored in the variable.
So if a variable is exported into an enviroment, if the shell ends, its still in the enviroment. Therefore it has changed to an enivorment variable. the command we use is

```
$ export varname
```

Basically, when the user logs in, there is an enviroment for the user. The enviroment is basically a set of variables

Here are some examples of those variables.

- HOME: full pathname of your home directory
- PATH: list of directories to search for commands
- USER: Your user name, also UID for user id
- SHELL: full pathname of your login shell
- PWD Current working directory
- HOSTNAME: current hostname of the system
- HISTSIZE: Number of commands to remember

- PS1: primary prompt
- ?: Return status of most recently executed command
- $: Process id of current process.

2.2 The PATH Variable

One variable that is quite useful is the PATH variable. The path variable lists a set of directories. The shell finds commands in these directories.

2.3 Bash shell prompt

the prompt can be changed vua the PS1 shell variable

```
PS1="$USER > "
```

Special "PS1" shell variable settings:

- \w current working directory
- \h hostname
- \u username
- \d date
- \t time
- \a righ the "bell"

Example:

```
$ PS1="\u@\h \w \$ "
student@csci330 ~ $
```

3 Shell aliases

Allows you to assign a different name to a command

To check current aliases:

```
$ alias
```

to set alias:

```
$ alias ll="ls -al"
```

to remove alias

```
$ unalias ll
```

4 Keeping variables

Variables set on the command line end when the shell end.
What we can do is create a file and write our aliases there. After we write them to the file, we can run the command `source` on the file and run it as a shell script.

```
source aliases
```

Now, all the alias created in the file will be able to be used.

5 Shell History

The shell can remember commands that you have typed in. This is called the shell history
Commands in the shell history can be:

- re-called
- edited
- re-executed

The size of the history is set via shell variables

- per session HISTSIZE=500
- per user HISTFILESIZE=100

To view the history buffer:

```
history [-c] [count]
```

6 Command Sequence

allows series of commands all at one

```
date;pwd;ls
```

7 Command substiution

command surrounded by back quotes is run and replaced by its standard output

newlines in the output are replaced by spaces

```
ls -l `which passwd`
var=`whoami`;echo $var
```

We can also perform command substitution like this

```
$ (command)
```

Here is an example of that:

```
echo User $(whoami) is on $(hostname)
```

8 Output redirection

Syntax: command > file
sends command output to file, instead of the terminal

Examples:

```
ls > listing
cat listing > filecopy
```

NOTE: if “file” exists, it is overwritten

9 Input Redirection

Syntax:

```
command < file
```

command reads (takes input) from file instead of keyboard

Example:

```
tr a-z A-Z < listing
```

this translates all the lowercase characters from the file *listing* to uppercase.

9.1 Examples: Output / input

redirecting input and output

```
tr A-Z a-z < r.in > r.out
```

Output of command becomes input to next:

```
ls > /tmp/out.txt; wc< /tmp/out.txt
```

Eliminate the middleman by using the pipe command

```
ls | wc
```

10 Appending Output

Sytax

```
command >> file
```

This adds output of command at the end of file
if the file does not exist, the shell creates it

Examples:

```
date > usage-status
ls -l >> usage-status
du -s >> usage-status
```

10.1 Here Document

read unput for current source, uses << symbol
Syntax:

```
command << LABEL
```

This reads following lines until line starting with "LABEL"

Example

```
wc - l << done
> line one
> line two
> DONE
2
```

11 File Descriptor

- positive integer for every open file
- process tracks its open files with this number

    0 - standard input
    1 - standard output
    2 - standard error output

Note:-

bash can use file descriptor to refer to a file

12 Redirecting syntax

- Output
    > or 1> filename
        2> filename
- input
    < or 0<
- Combining outputs
    2>&1 or &> or >&

Example:

```
cat mouse > /tmp/out.txt 2>&1

cat mouse &> /tmp/out.txt
```