Using the bash Shell

Command Line Shortcuts - File Globbing

● Globbing is wildcard expansion:

* ***** - matches zero or more characters
* **?** - matches any single character
* **[0-9]** - matches a range of numbers
* **[abc]** - matches any of the character in the list
* **[^abc]** - matches all except the characters in the list
* Predefined character classes can be used

Command Line Shortcuts - The Tab Key

● Type **Tab** to complete command lines: For the command name, it will complete a command name. For an argument, it will complete a file name

```
$ xte<Tab>
$ xterm
$ ls myf<Tab>
$ ls myfile.txt
```

Command Line Shortcuts History

● **bash** stores a history of commands you've entered, which can be used to repeat commands

● Use **history** command to see list of "remembered" commands

```
$ history
14 cd /tmp
... output truncated ...
```

Command Line Expansion The tilde

● Tilde (~) refers to your home directory

```
$ cat ~/.bash_profile
```

Command Line Expansion - Commands and Braced Sets

```
$ echo "This system's name is $(hostname)"
This system's name is star11.star.com
```

● Brace Expansion: **{ }** Shorthand for printing repetitive strings

```
$ echo file{1,3,5}
file1 file3 file5
$ rm -f file{1,3,5}
```

Scripting Basics: Shell scripts are text files that contain a series of commands or statements to be executed. Shell scripts are useful for:

* Automating commonly used commands
* Performing system administration and troubleshooting
* Creating simple applications
* Manipulation of text or files

Creating Shell Scripts

Step 1: Use such as **vi** to create a text file containing commands

```
#!/bin/bash
```

● Comment your scripts!: Comments start with a **#**

Step 2: Make the script executable: `$ chmod u+x myscript.sh`

● To execute the new script:

* Place the script file in a directory in the executable path -OR-
* Specify the absolute or relative path to the script on the command line

Sample Shell Script

```
#!/bin/bash
# This script displays some information about your environment
echo "Greetings. The date and time are $(date)"
echo "Your working directory is: $(pwd)"
```

Shell Programming - Basics

Standard Input and Output

- Linux provides three I/O channels to Programs
  - * Standard input (STDIN) - keyboard by default (<)
  - * Standard output (STDOUT) - terminal window by default (>)
  - * Standard error (STDERR) - terminal window by default (2>)
  - * `&>` Redirect all output to file
- File contents are overwritten by default. `>>` appends.

Redirecting STDIN from a File

- Redirect standard input with **<**

```
$ tr 'A-Z' 'a-z' < .bash_profile
```

  * This command will translate the uppercase characters in `.bash_profile` to lowercase. Equivalent to:

```
$ cat .bash_profile | tr 'A-Z' 'a-z'
```

Scripting: Examples : 1. Branching

```
#!/bin/sh
if [ "$1" = "1" ]
then
   echo "The first choice is nice"
elif [ "$1" = "2" ]
then
   echo "The second choice is just as nice"
elif [ "$1" = "3" ]
then
   echo "The third choice is excellent"
else
   echo "I see you were wise enough not to choose"
   echo "You win"
fi
$ sh branch 1
The first choice is nice
```

2. looping

```
#!/bin/sh
fruitlist="Apple Pear Tomato Peach Grape"
for fruit in $fruitlist
do
   if [ "$fruit" = "Tomato" ] || [ "$fruit" = "Peach" ]
   then
      echo "I like ${fruit}es"
   else
      echo "I like ${fruit}s"
   fi
done
```

3. Multiplication

```
$vi multable.sh
echo "enter the value of n:"
read n
i=1
for((i=1;i<=10;i++))
do
echo " $n * $i = `expr $n \* $i`"
done
```