Auto-Complete:
By pressing the tab key, you can use a built-in feature of Bash to automatically complete commands and file names. You can save a ton of time and typing by doing this!

Command history:
During the current session, Bash keeps track of every command you've entered. By typing history or by using the up and down arrow keys, you can access this history. Ctrl+r and a search term can also be used to search your command history.

Redirecting output:
By using the > operator, you can direct a command's output to a file. ls > output.txt, for instance, will save the results of the ls command to a file with the name output.txt.

Pipes: You can pass the output of one command as the input to another command by using the | operator. To list all files in the current directory that contain the string "foo," use the ls | grep foo command.

Aliases: The alias command can be used to create aliases for frequently used commands. For instance, alias ll='ls -alF' creates an alias ll that provides a detailed list of every file in the current directory.

Brace expansion: This technique can be used to produce a variety of values or file names. As an illustration, the commands touch file1..5 will produce the files file1.txt, file2.txt, file3.txt, file4.txt, and file5.txt.

Command substitution: This technique lets you use the result of one command as an argument for another command.

If Bash scripts are not performing as planned, how can I debug them?

While debugging Bash scripts might be difficult, there are a number of ways that can be useful. One typical method is to activate debugging mode using the set -x command, which prints each command as it is executed. This can assist in locating the script's problem areas and revealing the values being utilized for each variable. Using echo statements to report out variable values at different points in the script can also be useful for identifying problems.

How can I improve the security and effectiveness of my Bash scripts? Avoiding pointless loops and commands and using built-in Bash commands and features whenever possible will help Bash scripts run more quickly. Bash scripts should appropriately evaluate user input, employ appropriate file permissions, and refrain from using hazardous commands or features in order to increase their security. Additionally, encapsulating routine operations in functions can lessen code duplication and enhance maintainability.

Resources:

- https://wiki.bash-hackers.org/scripting/basics
- https://www.shellscript.sh/
- https://www.linuxjournal.com/content/shell-scripting-and-security
- https://www.landoflinux.com/linux_bash_scripting_structure.html
- https://phoenixnap.com/kb/wp-content/uploads/2022/11/linux-commands-cheat-sheet-pdf.pdf
- https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/windows-commands
- https://oit.ua.edu/wp-content/uploads/2020/12/Linux_bash_cheat_sheet-1.pdf
- https://devhints.io/bash
- https://exercism.org/tracks/bash