



Lecture 2: Shell Tools and Scripting

[Shell Scripting](#)

[Shell Tools](#)

[Finding files](#)

[Finding code](#)

[Finding shell commands](#)

[Directory Navigation](#)


Shell Scripting

1. Strings in bash can be defined with ' and " delimiters but they are not **equivalent**.

```
foo=bar
echo "$foo"
# prints bar
echo '$foo'
# prints $foo
```

Quoting (Bash Reference Manual)

3.1.2 Quoting Quoting is used to remove the special meaning of certain characters or words to the shell. Quoting can be used to disable special treatment for special characters, to prevent reserved words from being recognized as such, and to prevent parameter expansion.

 https://www.gnu.org/software/bash/manual/html_node/Quoting.html

2. Special characters

- `$0` - Name of the script
- `$1` to `$9` - Arguments to the script. \$1 is the first argument and so on.
- `$@` - All the arguments
- `$#` - Number of arguments
- `$?` - Return code of the previous command
- `$$` - Process Identification number for the current script
- `!!` - Entire last command, including arguments.
- `$_` - Last argument from the last command. If you are in an interactive shell, you can also quickly get this value by typing Esc followed by .
- `||` - OR logical operator. In a test construct, the `||` operator causes a return of 0 (success) if *either* of the linked test conditions is true.
- `&&` - AND logical operator. In a test construct, the `&&` operator causes a return of 0 (success) only if *both* the linked test conditions are true.
- `;` - Commands can also be separated within the same line using a semicolon


3. Command substitution & process substitution

A command substitution (`$(...)`) will be replaced by the **output** of the command.

A process substitution (`<(...)`) will be replaced by a filename from which the output of the command may be read.

Difference between subshells and process substitution

A command substitution (`$(...)`) will be replaced by the output of the command, while a process substitution (`<(...)`) will be replaced by a filename from which the output of the command

 <https://unix.stackexchange.com/questions/393349/difference-between-subshells-and-process-substitution>



4. Shell globbing

- Wildcards - You can use `?` and `*` to match one or any amount of characters respectively.
- Curly braces `{ }` - Whenever you have a common substring in a series of commands you can use curly braces for bash to expand this automatically.

```
convert image.{png,jpg}
# Will expand to
convert image.png image.jpg

cp /path/to/project/{foo,bar,baz}.sh /newpath
# Will expand to
cp /path/to/project/foo.sh /path/to/project/bar.sh /path/to/project/baz.sh /newpath
```



Writing bash scripts can be tricky and unintuitive. There are tools like [shellcheck](#) that will help you find out errors in your sh/bash scripts.

Shell Tools

Finding files

```
# Find all directories named src
find . -name src -type d
# Find all python files that have a folder named test in their path
find . -path '**/test/**/*.py' -type f
# Find all files modified in the last day
find . -mtime -1
# Find all zip files with size in range 500k to 10M
find . -size +500k -size -10M -name '*.tar.gz'
```



[fd](#) is a simple, fast and user-friendly alternative to find. It offers some nice defaults like colored output, default regex matching, Unicode support and it has in what my opinion is a more intuitive syntax.

Finding code

`grep` has many flags that make it a very versatile tool.

- `-C` - getting *Context* around the matching line
- `-v` - *inverting* the match, i.e. print all lines that do not match the pattern
- `-R` - *recursively* go into directories and look for text files for the matching string



`grep -R` can be improved in many ways, such as ignoring `.git` folders, using multi CPU support. So there has been no shortage of alternatives developed, including [ack](#), [ag](#) and [rg](#).

Finding shell commands

The `history` command will let you access your shell history programmatically. If we want to search there we can pipe that output to `grep` and search for patterns. `history | grep find` will print commands with the substring “find”.

In most shells you can make use of `Ctrl+R` to perform backwards search through your history

Directory Navigation

Finding frequent and/or recent files and directories can be done through tools like `fasd`. `Fasd` ranks files and directories by `freccency`, that is, by both frequency and recency.



More complex tools exist to quickly get an overview of a directory structure `tree`, `broot` or even full fledged file managers like `nnn` or `ranger`