Pattern Matching in the Shell

Globbing

Globbing is a simplified form of pattern matching used by the shell (e.g., bash) primarily for matching file names. It's distinct from, but related to, regular expressions. - Commonly used with commands like `echo`, `ls`, and `rm`. - It can also be used within control structures such as `case`. - Faster and simpler than full regex. - Only matches filename components (no slashes).

Special Characters and Syntax

Symbol	Meaning
`*`	Matches any sequence of zero or more characters.
`?`	Matches exactly one character.
`[...]`	Matches any single character enclosed in the brackets.
`[a-z]`	Matches characters in the specified range (`a` to `z`).
`[!...]`	Matches a single character not in the specified set (negation).

- Use `-` in brackets to denote ranges. To include the `-` itself, escape ambiguity by placing it at the end.
- Directory separator `/` is not matched by `*` or `?`.
- Filenames that start with `.` (dot files) are not matched by `*` or `?` unless the pattern explicitly starts with `.`, e.g., `.??*` matches hidden files of at least three characters.

Globbing Exceptions

1. Globs never match slashes (`/`)
    - This prevents expensive recursive directory traversals.
2. File and directory names starting with `.` (dot files) are hidden and not matched by wildcards unless explicitly specified.

Globbing Examples

Pattern	Description	Match
`*`	All files except hidden ones.	`foo`, `bar.txt`, not `.hidden`
`?.txt`	One character followed by `.txt`	`a.txt`, not `ab.txt`
`[abc]*`	Files starting with `a`, `b`, or `c`.	`apple`, `cat`, not `dog`
`[!abc]*`	Files not starting with `a`, `b`, or `c`.	`dog`, not `apple`
`*.bash`	Files ending with `.bash`.	`script.bash`
`.??*`	Hidden files with name length ≥ 3.	`.xrc`, not `.x`

Example Directory: Contains files foo, bar, .hidden, a.txt, a-b.txt

Command	Matches
echo *	foo, bar, a.txt, a-b.txt
echo .*	.hidden, .., .
echo ?.txt	None (requires 1-char name)
echo [a-c]*	bar
echo [!abc]*	Anything not starting a/b/c

Globbing in Directory Hierarchies

- Directory matching needs explicit structure:
  - `*/foo*` only finds `foo*` in direct subdirectories.
  - `*/*/foo*` required for two-level search.

Globbing Exclusion

You can use bracket negation to exclude specific starting characters.

Example:

```
echo [!abc.]*
```

Matches files that do not begin with `a`, `b`, `c`, or `.`.

---

2. Shell I/O Redirection

Standard File Descriptors

Descriptor	Description
0	Standard Input
1	Standard Output
2	Standard Error

Redirection Syntax

Syntax	Meaning
`>`	Redirect `stdout` to a file (overwrite).
`>>`	Redirect `stdout` and append to file.
`2>&1`	Redirect `stderr` to where `stdout` points.
`3< file`	Open file on descriptor 3 for reading.
`3> file`	Open file on descriptor 3 for writing.
`3<> file`	Open for reading and writing on descriptor 3.

Here Documents

A “here document” is a redirect that includes text directly in the script or command.

Example:

```
cat <<EOF
line one
line two
EOF
```

This feeds the lines into `cat` as `stdin`.

- Supports variable substitution unless quoted:

```
cat << 'EOF' # prevents variable expansion
cat << EOF   # expands variables
```

Advanced Redirection Examples

```
command 3>&1 1>tempfile
```

- Redirects file descriptor 3 to the current `stdout`, and `stdout` to a file.

---

3. Shell Commands and Scripting

Exit and Return

Command	Effect
`exit`	Terminates shell (or script).
`exit 1`	Exits with status code 1.
`return`	Exits from a shell function only.

- `exit` exits the shell completely, but `return` is confined to functions.
- You can check the exit status using `$?`.

Shell Functions vs Shell Scripts

Two approaches:

Shell Function

```
g() {
  grep "$@"
}
```

- Defined in `.profile` or directly in the shell.
- Lightweight and executed in current shell.
- Faster (no new process)

Shell Script

g file:

```
#!/bin/bash
grep "$@"
```

- Saved in file in a directory listed in `$PATH`.
- Executed in a new process.
- More portable and global.

Feature	Function	Script
Scope	Local to shell	Global in environment
Overhead	Low	High (new process)
Portability	Low	High
Speed	Fast for small tasks	Better for large tasks

Aliases

- Used for simple command substitution.

```
alias g='grep'
```

- Not suitable for complex logic; use functions instead.

---

4. Regular Expressions

Design Philosophy

Regex (regular expressions) define patterns to match strings. Used with tools like grep, sed, awk, Python, etc. - A little language tailored to string pattern matching. - Variants exist because different tool authors chose different syntaxes.

Extended Regular Expressions (ERE)

Use grep -E or egrep to avoid these issues.

Core Syntax and Operators

Operator	Description
`.`	Any character except newline
`*`	0 or more repetitions
`+`	1 or more repetitions
`?`	0 or 1 occurrence
`|`	Alternation (OR) between regex
`()`	Grouping (changes precedence)
`{n}`	Exactly n occurrences
`{n,m}`	Between n and m repetitions
`^`	Start of line anchor
`$`	End of line anchor
`[]`	Bracket expressions (character classes)

Bracket expressions

Syntax	Description
`[abc]`	Match ‘a’, ‘b’, or ‘c’
`[a-z]`	Range: a to z
`[^abc]`	Negate: anything except a, b, or c
`[[:alpha:]]`	Match alphabetical letters (locale-aware)
`[.]`, `[-]`, `[]]` tricks	Special syntax rules to include symbols

Special character handling inside `[]`:

- `-` denotes range unless at start or end
- `]` must be escaped or placed first
- `^` must be first character to negate

Examples

Pattern	Matches Example
`abc`	Only the string abc
`a.b`	a followed by any character, then b
`a*`	zero or more a’s
`a+`	one or more a’s
`(ab|cd)+`	ab or cd, repeated
`^xyz$`	entire line must be `xyz`
`^(.)(.)(.).\3\2\1$`	Matches six-character palindromes.

Escape Sequences

Backslashes are used to escape special characters. - Must escape metacharacters: `\* \. \( \)` etc. - Caution: Shell may interpret before `grep` does.

To match `\`, you’ll often need `\\\\`

Basic Regular Expressions (BRE)

Used with `grep` without `-E`.

Differences from ERE: - Metacharacters like `+`, `?`, `{}` are NOT special. - Use `\(...\)` for grouping. - Use `\{n,m\}` for repetition. - `|` is not supported directly.

Backreferences (BRE only)

Syntax	Description
`\1`	Refers to first captured group.
`\2`	Refers to second captured group.

Examples

Pattern	Description
`\(abc\)\1`	Matches ‘abcabc’
`\(^a.*b$\)`	Entire line starting with a, ending with b
`^\(.\)\(.\)\2\1$`	Matches 4-character palindromes.

Performance Note: Backreferences are slow and non-regular — avoid when possible.

Common Pitfalls

- Regular expressions with just a backslash (`\`) are invalid.
- Quoted expressions inside the shell need escaping.

---

5. Emacs

Philosophy

- Keyboard-driven efficiency.
- Emphasis on not taking hands off keyboard.
- Modular via modes and the mini-buffer.

Cursor and Region Operations

I/O

Command	Action
`C-x C-f`	Open file
`C-x C-s`	Save file
`C-g`	Cancel current command

Mark, Region, and Copy

Command	Action
`C-SPC` or `C-@`	Set mark at cursor (start selection)
`M-w`	Copy selected region to kill-ring
`C-w`	Cut (aka “kill”) selected region
`C-y`	Yank (paste) last kill-ring
`M-y`	Paste earlier entries in kill-ring
`C-x C-x`	Exchange point and mark

The Kill Ring

- Stores multiple text entries from kills.
- Cyclic navigation with `M-y` after `C-y`.

Buffer and Window Management

- Buffers: Independent in-memory views (files, outputs, shell, etc.)
- Windows: Viewports into buffers

Command	Description
`C-x b`	Switch buffer.
`C-x C-b`	List all buffers.
`C-x o`	Switch windows.
`C-x 2`	Split window horizontally.
`C-x 3`	Split window vertically.
`C-x 0`	Close current window.
`C-x 1`	Maximize current window.

Emacs Modes

- Major modes (e.g., Fundamental, Dired) tailor behavior to the file type or buffer.
- Minor modes add auxiliary behavior (e.g., Line numbers).

Use `C-h m` to describe current modes and key bindings.

Accessing Help

Command	Description
`C-h k`	Describe key binding.
`C-h m`	Describe current mode.
`C-h i`	Info documentation browser.

Meta key (`M-`) is typically `Alt` or `Esc` key.

Mini-buffer Operations

Executes internal commands or inputs.

- `M-x`: Execute Emacs command.
- `M-:`: Evaluate Emacs Lisp.
- `M-!`: Run shell command.
- `M-|`: Run shell command with region as input.

Examples:

- `M-! date`: Run shell date command
- `M-| sort`: Sort selected region

Key behaviors:

- Uses same movement/copy/edit commands
- Allows evaluation of Emacs Lisp
