Bash Expansions

This handout covers essential expansions in Bash, allowing dynamic command rewriting and powerful scripting capabilities. It includes tilde, variable, filename expansions, word splitting, quoting, command substitution, and escaping special characters.

---

1. Introduction to Bash Expansions

Shell expansions allow Bash to dynamically rewrite commands before execution, enabling:

- Simplified syntax for complex operations.
- Dynamic variable replacement.
- Powerful file selection mechanisms.

Expansions happen **before** the command is executed.

---

2. Tilde Expansion (`~`)

The tilde symbol expands to your home directory path.

- Basic usage:

```
echo ~
# Output: /home/username
```

- Current directory expansion (`~+`):

```
echo ~+
# Outputs the current working directory
```

---

3. Variable and Shell Parameter Expansion

Variable Expansion (`$`)

Replace a variable with its stored value.

Syntax:

```
echo "${HOME}"
```

- **Always prefer curly braces `{}` for clarity**.

Example

```
echo "Home is: ${HOME}"
```

Shell Parameter Expansion

Perform simple operations on variable strings.

- **Length of variable (`#`)**:

```
echo "${#HOME}"
# Outputs length of HOME variable (e.g., 11)
```

- **Substring extraction**:

```
echo "${HOME:0:5}"
# Outputs first 5 characters
```

- **Replace string in variable (`//`)**:

```
echo "${HOME//home/users}"
# Replaces all occurrences of 'home' with 'users'
```

---

3. Filename Expansion (Globbing)

Use wildcards for matching filenames.

- **** `*` **matches zero or more characters**:

```
ls *.txt
```

- `?` **matches exactly one character**:

```
ls file?.txt
# Matches: file1.txt, fileA.txt, but not file12.txt
```

- **Square brackets** `[ ]` **match one character within a range**:

```
ls file[0-9].txt
# Matches: file1.txt, file2.txt; not fileA.txt
```

---

4. Word Splitting

Bash splits commands into separate arguments based on whitespace (spaces, tabs, newlines).

Example:

```
touch file1.txt file2.txt
```

Bash splits this into:

- Command → `touch`

- Arguments → `file1.txt`, `file2.txt`

To disable word splitting (e.g., for filenames with spaces), use quotes:

```
touch "my file.txt"
```

---

5. Quotes in Bash Commands

Quotes affect expansions and word splitting:

Quote Type	Variable Expansion	Filename Expansion	Tilde Expansion	Word Splitting
No quotes	✅	✅	✅	✅
Single quotes (' ')	❌	❌	❌	❌
Double quotes (" ")	✅	❌	❌	❌

- **Best practice**: Use the most restrictive quoting style (`' '`) possible.

Example:

```
echo '$HOME/*.txt'  # No expansions (literal output)
echo "$HOME/*.txt"  # Expands HOME, no filename expansion
echo ~/*.txt        # Expands HOME and filenames
```

---

6. Dangers of Filename Expansion

Globbing ↔ Matching filenames using wildcard characters (e.g., `*`).

- Example (list all `.txt` files):

```
ls *.txt
```

⚠️ Potential dangers:

If filenames are misunderstood as command arguments:

- Example dangerous scenario:

```
rm *
```

> *Note: Be careful with wildcards! A file named -rf could be interpreted as a command parameter when using `rm`, leading to unintended deletions.*

Safe approach:

Prefix filenames explicitly to avoid misinterpretation:

```
rm ./*.txt
```

---

7. Command Substitution (`$(command)`)

Execute commands and capture their outputs directly into another command.

Syntax:

```
echo "Today is $(date)"
```

Example (file count):

```
echo "There are $(ls | wc -l) files."
```

> *Note: Prefer $() over the older `command` syntax.*

---

8. Escaping Characters (`\`)

Use backslash (`\`) to prevent special characters from being interpreted by Bash.

- Example (filename with space):

```
touch my\ file.txt
```

- Escaping quotes within quotes:

```
echo "Hello \"World\""
```

- Single quotes disable escaping entirely:

```
echo 'This is a $variable'
# Outputs literally: This is a $variable
```

Important Note on Single Quotes (`''`):

- All expansions and escapes (`\`) are disabled.
- Use double quotes `" "` when expansion is required.

📌 Summary of Key Commands

Expansion Type	Example	Description
Tilde (~)	`echo ~`	Expands to home directory (`/home/user`)
Variable (`$`)	`echo "${HOME}"`	Inserts variable value

Shell Parameter (`${}`)	`echo "${#HOME}"`	Performs string operations
Filename Globbing (`*`)	`ls *.txt`	Matches filenames using wildcards
Command substitution	`echo "$(date)"`	Inserts command output
Word Splitting	`touch file1 file2`	Splits command based on whitespace
Escaping (`\`)	`touch my\ file.txt`	Escapes special characters