**~ (tilde) operator**

**In shell scripting, the ~ (tilde) operator is often used for pattern matching in some shell environments like bash with extended globbing, but for checking patterns like digits or alphabets, we typically use [[ ]] with regular expressions.**

### **1. Check if a String Contains Only Digits**

**#!/bin/bash**

**read -p "Enter a string: " input**

**if [[ $input =~ ^[0-9]+$ ]]; then**

**echo "The string contains only digits."**

**else**

**echo "The string does not contain only digits."**

**fi**

### **1. ^ (Caret)**

* **The caret symbol ^ is an anchor in regular expressions that asserts the position at the start of a line or string.**
* **This means that whatever pattern comes after ^ must match from the very beginning of the string.**

### **2. [0-9] (Character Class)**

* **The square brackets [ ] denote a character class.**
* **[0-9] is a character class that matches any single digit from 0 to 9.**
  + **0-9 is a range, which is shorthand for 0123456789.**

### **3. + (One or More)**

* **The + symbol is a quantifier in regular expressions.**
* **It specifies that the preceding element (in this case, [0-9]) must occur one or more times.**

### **4. $ (Dollar Sign)**

* **The dollar sign $ is another anchor in regular expressions, similar to ^.**
* **However, $ asserts the position at the end of a line or string.**
* **This means that the pattern must match up to the very end of the string.**

### **Putting It All Together: ^[0-9]+$**

**When you combine all these components, the regular expression ^[0-9]+$ does the following:**

* **^ asserts that the match must start at the beginning of the string.**
* **[0-9]+ matches one or more digits in a row.**
* **$ asserts that the match must end at the very end of the string.**

### **2. Check if a String Contains Only Alphabets**

**#!/bin/bash**

**read -p "Enter a string: " input**

**if [[ $input =~ ^[a-zA-Z]+$ ]]; then**

**echo "The string contains only alphabets."**

**else**

**echo "The string does not contain only alphabets."**

**fi**

* **[a-zA-Z]+ matches one or more alphabetic characters (both lowercase and uppercase).**

### **3. Check if a String Contains Both Digits and Alphabets**

**#!/bin/bash**

**read -p "Enter a string: " input**

**if [[ $input =~ ^[a-zA-Z0-9]+$ ]]; then**

**echo "The string contains only digits and alphabets."**

**else**

**echo "The string contains other characters apart from digits and alphabets."**

**fi**

* **[a-zA-Z0-9]+ matches one or more alphanumeric characters.**

**More examples of regular expression patterns commonly used in shell scripting, along with explanations and practical use cases**

### **^[a-zA-Z0-9\_]+$**

* **Pattern: Matches a string that contains only alphanumeric characters and underscores.**
* **Explanation:**
  + **^: Start of the string.**
  + **[a-zA-Z0-9\_]: Matches any letter (uppercase or lowercase), digit (0-9), or underscore (\_).**
  + **+: Ensures one or more occurrences of the previous character class.**
  + **$: End of the string.**
* **Examples:**
  + **"user\_name123": Match.**
  + **"username!": No match (contains an exclamation mark).**
  + **"123": Match (contains only digits).**

### **2). ^[0-9]{3}-[0-9]{2}-[0-9]{4}$**

* **Pattern: Matches a string in the format of a Social Security Number (SSN) in the U.S., like 123-45-6789.**
* **Explanation:**
  + **^: Start of the string.**
  + **[0-9]{3}: Matches exactly three digits.**
  + **-: Matches the hyphen character.**
  + **[0-9]{2}: Matches exactly two digits.**
  + **-: Matches the hyphen character.**
  + **[0-9]{4}: Matches exactly four digits.**
  + **$: End of the string.**
* **Examples:**
  + **"123-45-6789": Match.**
  + **"12-345-6789": No match (first part does not have three digits).**
  + **"123456789": No match (missing hyphens).**

### **3). ^[A-Z]{1}[a-z]{2,}$**

* **Pattern: Matches a string that starts with one uppercase letter, followed by at least two lowercase letters.**
* **Explanation:**
  + **^: Start of the string.**
  + **[A-Z]{1}: Matches exactly one uppercase letter.**
  + **[a-z]{2,}: Matches at least two lowercase letters (but can match more).**
  + **$: End of the string.**
* **Examples:**
  + **"John": Match.**
  + **"Jo": No match (only one lowercase letter after the uppercase).**
  + **"john": No match (does not start with an uppercase letter).**
  + **"J123": No match (contains digits).**

### **4). ^([a-z0-9\.\_%+-]+)@([a-z0-9.-]+)\.([a-z]{2,})$**

* **Pattern: Matches a valid email address.**
* **Explanation:**
  + **^: Start of the string.**
  + **([a-z0-9\.\_%+-]+): Matches the local part of the email (letters, digits, dots, underscores, percentages, pluses, minuses).**
  + **@: Matches the "@" symbol.**
  + **([a-z0-9.-]+): Matches the domain part of the email (letters, digits, dots, hyphens).**
  + **\.: Matches the dot before the domain extension.**
  + **([a-z]{2,}): Matches the domain extension (at least two letters).**
  + **$: End of the string.**
* **Examples:**
  + **"user@example.com": Match.**
  + **"user.name@domain.co": Match.**
  + **"user@domain": No match (missing domain extension).**
  + **"user@domain.c": No match (domain extension too short).**

### **5.) ^([0-9]{1,3}\.){3}[0-9]{1,3}$**

* **Pattern: Matches an IPv4 address format.**
* **Explanation:**
  + **^: Start of the string.**
  + **([0-9]{1,3}\.){3}: Matches three groups of one to three digits followed by a dot.**
  + **[0-9]{1,3}: Matches the last group of one to three digits.**
  + **$: End of the string.**
* **Examples:**
  + **"192.168.1.1": Match.**
  + **"255.255.255.0": Match.**
  + **"192.168.1.": No match (trailing dot without digits).**
  + **"999.999.999.999": Match (but not a valid IP address in the real world).**

### **6.) ^[^a-zA-Z0-9]+$**

* **Pattern: Matches a string that contains only special characters (no letters or digits).**
* **Explanation:**
  + **^: Start of the string.**
  + **[^a-zA-Z0-9]: Matches any character that is not a letter or digit (^ inside a character class negates it).**
  + **+: Ensures one or more occurrences of special characters.**
  + **$: End of the string.**
* **Examples:**
  + **"@#$%^&\*": Match.**
  + **"123abc!": No match (contains letters and digits).**
  + **"!!!": Match (contains only special characters).**

### **7.) ^[A-Z]{1}[a-zA-Z]\*\d{0,}$**

* **Pattern: Matches a string that starts with an uppercase letter, followed by any number of alphabetic characters, and optionally ends with digits.**
* **Explanation:**
  + **^: Start of the string.**
  + **[A-Z]{1}: Matches exactly one uppercase letter.**
  + **[a-zA-Z]\*: Matches zero or more alphabetic characters.**
  + **\d{0,}: Matches zero or more digits at the end.**
  + **$: End of the string.**
* **Examples:**
  + **"JohnDoe123": Match.**
  + **"A1": Match.**
  + **"john123": No match (does not start with an uppercase letter).**

**4) shell script to validate a 16 digit credit card number.**

**#!/bin/bash**

**# Prompt the user to enter a 16-digit number**

**read -p "Enter a 16-digit number: " input\_number**

**# Check if the input matches exactly 16 digits**

**if [[ $input\_number =~ ^[0-9]{16}$ ]]; then**

**echo "Valid 16-digit number."**

**else**

**echo "Invalid input. Please enter a 16-digit numeric value."**

**fi**

**^: Asserts the start of the string.**

**[0-9]{16}: Matches exactly 16 digits.**

**$: Asserts the end of the string.**

**5) shell script that checks for a pattern where:**

1. **The string starts with exactly five alphabetic characters.**
2. **This is followed by exactly four digits.**
3. **The string ends with exactly one alphabets**

**#!/bin/bash**

**# Prompt the user to enter the pattern**

**read -p "Enter the pattern (e.g., abcde1234x): " input\_pattern**

**# Check if the input matches the required pattern**

**if [[ $input\_pattern =~ ^[a-zA-Z]{5}[0-9]{4}[a-zA-Z]$ ]]; then**

**echo "Valid pattern."**

**else**

**echo "Invalid pattern. Please enter a string that matches '#####0000#' where # is an alphabet and 0 is a digit."**

**fi**

**6) shell script to check if a string is a valid Social Security Number (SSN), you need to validate the format. A typical SSN format is XXX-XX-XXXX, where:**

* **XXX: Three digits**
* **XX: Two digits**
* **XXXX: Four digits**

### 

**#!/bin/bash**

**# Prompt the user to enter the SSN**

**read -p "Enter the Social Security Number (e.g., 123-45-6789): " ssn**

**# Check if the input matches the SSN format**

**if [[ $ssn =~ ^[0-9]{3}-[0-9]{2}-[0-9]{4}$ ]]; then**

**echo "Valid SSN format."**

**else**

**echo "Invalid SSN format. Please enter a string that matches 'XXX-XX-XXXX' where X is a digit."**

**fi**

### **Explanation**

* **Pattern Explanation:**
  + **^[0-9]{3}: Matches exactly three digits at the start.**
  + **-: Matches the hyphen character.**
  + **[0-9]{2}: Matches exactly two digits.**
  + **-: Matches the hyphen character.**
  + **[0-9]{4}$: Matches exactly four digits at the end**

**Common Patterns and Use Cases in Jenkins**

**For validating strings**

1. **Feature Branch Naming:**

* **Pattern: ^feature/[a-zA-Z0-9-\_]+$**
* **Use Case: Ensure that feature branches follow a specific naming convention, e.g., feature/new-ui or feature/xyz-123.**

### **Tag Names**

* **Pattern: ^v?[0-9]+\.[0-9]+\.[0-9]+$**
* **Use Case: Ensures that tag names follow semantic versioning, optionally prefixed with a "v", e.g., v2.1.0 or 2.1.0.**

### **. Build Numbers**

* **Pattern: ^[0-9]+$**
* **Use Case: Validates that the input is a positive integer, typically used for build numbers or version numbers.**

### **4.) Date in YYYY-MM-DD Format**

* **Pattern: ^[0-9]{4}-[0-9]{2}-[0-9]{2}$**
* **Use Case: Ensures that the input follows the date format YYYY-MM-DD, e.g., 2024-08-24**

### **5) IP Addresses**

* **Pattern: ^([0-9]{1,3}\.){3}[0-9]{1,3}$**
* **Use Case: Ensures the input matches a standard IPv4 address format, e.g., 192.168.0.1.**

### **6). Usernames**

* **Pattern: ^[a-zA-Z0-9.\_-]{3,16}$**
* **Use Case: Validates usernames to be between 3 and 16 characters long, and can include alphanumeric characters, dots, underscores, or hyphens.**

**7)Docker Image Tags**

* **Pattern: ^[a-zA-Z0-9.\_-]+(:[a-zA-Z0-9.\_-]+)?$**
* **Use Case: Ensures Docker image tags are valid, allowing optional version suffixes, e.g., myapp:latest or myapp**