Bash 치트시트

기본 및 변수

- `echo <string>`: 문자열을 출력합니다. `-e` 옵션을 사용하면 이스케이프 시퀀스(n, t 등)를 해석할 수 있습니다.
- `printf <format> <args>`: C 언어 스타일의 서식 지정자를 사용해 형식화된 문자열을 출력합니다. (예: `printf "이름: %s\nID: %d\n" "Jules" 123`)
- `read <var>`: 사용자의 입력을 받아 변수에 저장합니다. `-p "Prompt"`로 안내 문구를 지정하거나, `-s`로 입력값을 화면에 표시하지 않고 숨길 수 있습니다.
- 변수 선언: `foo=bar` (등호 앞뒤에 공백이 없어야 합니다. `foo = bar`는 `foo` 프로그램을 호출하는 것으로 해석됩니다.)
- 변수 사용: `$foo` 또는 `${foo}` 형식을 사용합니다.
- 따옴표 차이:
  - `' '` (작은따옴표): 문자열 그대로를 처리합니다. (`'$foo'` -> `$foo`)
  - `" "` (큰따옴표): 변수 값을 치환하여 처리합니다. (`"$foo"` -> `bar`)
- 명령어 치환: `output=$(command)` 또는 `command` 형식을 사용해 명령어의 실행 결과를 변수나 위치에 대체합니다.
- 프로세스 치환: `<(command)`는 명령어 결과를 임시 파일에 배치하고 해당 파일 경로로 대체합니다. (`diff <(ls foo) <(ls bar)`)
- 환경 변수: `export VAR="value"` 명령으로 현재 셸과 하위 프로세스에서 사용할 환경 변수를 정의합니다.
- 특수 변수:
- `$0`: 실행 중인 스크립트의 이름입니다.
- `$1` `$9`: 스크립트나 함수에 전달된 위치 기반 인자들입니다.
- `$#`: 전달된 전체 인자의 개수입니다.
- `$*`: 모든 인자를 하나의 문자열로 묶어서 취급합니다.
- `$@`: 모든 인자를 각각 별개의 문자열로 취급합니다.
- `$?`: 마지막으로 실행된 명령어의 종료 상태 코드입니다. (0은 성공/참, 0 이외는 에러/거짓)
- `$$`: 현재 실행 중인 셸의 프로세스 ID(PID)입니다.
- `!!`: 인수를 포함하여 마지막 명령 전체를 다시 실행합니다. (예: `sudo !!`)
- `$_`: 마지막 명령의 마지막 인자입니다. (대화형 셸에서 `Esc` + `.` 으로 입력 가능)

조건문

- `if [ condition ]; then ... fi`: 가장 기본적인 `if` 문입니다.
- `if [[ condition ]]; then ... fi`: 확장된 조건문으로, 정규 표현식 지원 등 보다 강력하고 안전한 기능을 제공합니다.
- `case $variable in pattern1) ... ;; pattern2) ... ;; *) ... ;; esac`: 여러 패턴을 비교할 때 사용하는 `case` 문입니다.

조건 표현식 (`[[ ... ]]` 내부)

- 문자열 비교: `[[ "$a" == "$b" ]]`, `[[ "$a" != "$b" ]]`, `[[ -z "$a" ]]` (빈 문자열 여부), `[[ -n "$a" ]]` (비어 있지 않은 문자열 여부)
- 정규 표현식 매칭: `[[ "$str" =~ $regex ]]`
- 숫자 비교: `(( a == b ))`, `(( a > b ))` 등 산술 비교를 사용합니다. (기존의 `[ $a -eq $b ]` 방식보다 권장됩니다.)
- 파일 검사:
  - `-e`: 파일이나 디렉토리가 존재하는지 확인합니다.
  - `-f`: 일반 파일인지 확인합니다.
  - `-d`: 디렉토리인지 확인합니다.
  - `-r`, `-w`, `-x`: 각각 읽기, 쓰기, 실행 권한이 있는지 확인합니다.
- 논리 연산: `&&` (AND), `||` (OR), `!` (NOT)

반복문

- `for var in item1 item2 ...; do ... done`: 목록의 각 항목에 대해 반복합니다.
- `for var in {1..10}; do ... done`: 중괄호 확장을 사용해 범위 기반 반복을 수행합니다.
- `for var in $(command); do ... done`: 명령어 실행 결과의 각 단어에 대해 반복합니다.
- C 스타일 `for` 루프: `for (( i=0; i<10; i++ )); do ... done`
- `while condition; do ... done`: 조건이 참인 동안 계속 반복합니다.
  - `while read -r line; do ... done < file.txt`: 파일의 내용을 한 줄씩 순차적으로 읽어 처리합니다.
- `until condition; do ... done`: 조건이 참이 될 때까지(거짓인 동안) 반복합니다.
- `break`: 루프를 즉시 종료합니다.
- `continue`: 현재 반복을 건너뛰고 다음 반복을 시작합니다.

함수

- 정의 방법 (두 가지):

```
function my_func { ... }
my_func() { ... }
```

- 호출: `my_func arg1 arg2` (괄호 없이 이름과 인자를 나열합니다.)
- 인자 접근: `$1`, `$2`, `$@` 등 스크립트와 동일한 방식으로 접근합니다.
- `return <number>`: 0에서 255 사이의 정수 종료 코드를 반환합니다.
- 실행 결과 캡처: `result=$(my_func)` 형식을 사용합니다.
- `local <var>`: 함수 내부에서만 유효한 지역 변수를 선언합니다.

배열 (Arrays)

- 선언 및 할당: `arr=("apple" "banana" "cherry")`
- 요소 접근: `${arr[0]}` (첫 번째 요소), `${arr[1]}` (두 번째 요소)
- 모든 요소 참조: `${arr[@]}`
- 모든 인덱스 참조: `${!arr[@]}`
- 배열의 크기(길이): `${#arr[@]}`
- 요소 추가: `arr+=("new item")`
- 연관 배열 (키-값 쌍):
  - `declare -A my_map` (먼저 명시적으로 선언해야 합니다.)
  - `my_map["key1"]="value1"`
  - `echo ${my_map["key1"]}`

문자열 조작

- 문자열 길이: `${#string}`
- 부분 문자열 추출: `${string:position:length}` (예: `${string:0:5}`)
- 패턴 제거 (앞부분):
  - `${string#pattern}`: 앞에서부터 가장 짧게 일치하는 패턴을 제거합니다.
  - `${string##pattern}`: 앞에서부터 가장 길게 일치하는 패턴을 제거합니다.
- 패턴 제거 (뒷부분):
  - `${string%pattern}`: 뒤에서부터 가장 짧게 일치하는 패턴을 제거합니다.
  - `${string%%pattern}`: 뒤에서부터 가장 길게 일치하는 패턴을 제거합니다.
- 치환:
  - `${string/pattern/replacement}`: 처음으로 일치하는 패턴을 치환합니다.
  - `${string//pattern/replacement}`: 일치하는 모든 패턴을 치환합니다.

셸 확장 (Expansions)

- `~`: 현재 사용자의 홈 디렉토리로 확장됩니다.
- `*`, `?`, `[...]`: 파일 경로를 확장하는 글로빙(Globbing) 패턴입니다.
- `{a,b,c}`: 중괄호 확장으로 여러 문자열 조합을 생성합니다. (예: `touch file_{1..3}.txt`)
- `$(...)`: 명령어의 실행 결과로 치환됩니다.
- `$((...))`: 내부에 기술된 산술 연산을 수행하고 그 결과로 치환됩니다.

스크립팅 팁과 모범 사례

- `set -e`: 스크립트 도중 명령어가 실패하면 즉시 실행을 중단합니다.
- `set -u`: 선언되지 않은 변수를 사용하려고 할 때 에러를 발생시킵니다.
- `set -o pipefail`: 파이프라인에서 연결된 명령어 중 하나라도 실패하면 전체 결과로 실패를 반환합니다.
- `trap 'cleanup' EXIT`: 스크립트가 종료될 때(정상 종료 및 중단 포함) 특정 함수나 명령어를 실행합니다.
- `getopts`: 스크립트에 전달된 옵션과 플래그를 체계적으로 파싱합니다.
- `readlink -f "$0"`: 현재 실행 중인 스크립트 파일의 실제 절대 경로를 얻습니다.
- `shebang`: 스크립트의 첫 줄에 `#!/bin/bash` 또는 `#!/usr/bin/env bash`를 작성하여 실행할 인터프리터를 지정합니다.

Google Style Guide

기본 원칙

- 쉘 선택: 실행 파일에는 `Bash`만 허용 (`#!/bin/bash`).
- 언제 사용하나: 데이터 조작이 적고 다른 유틸리티를 주로 호출하는 작은 유틸리티나 래퍼 스크립트.
- 복잡도: 스크립트가 100줄을 넘거나 제어 흐름이 복잡해지면 파이썬 등 구조화된 언어로 재작성 권장.

포매팅 (Formatting)

- 들여쓰기: 공백 2개 (탭 금지).
- 줄 길이: 최대 80자.

- 제어 구조: `; then`과 `; do`를 `if`/`for`/`while`과 같은 줄에 배치.
- 파이프라인: 한 줄에 다 안 들어가면 줄당 하나의 파이프로 나누고 2칸 들여쓰기.
- 변수 확장: `"$var"`보다 `"${var}"` 형식을 선호하며, 항상 따옴표를 사용할 것.

명명 규칙 (Naming)

- 함수: `lower_with_under()`, 패키지는 `::`로 구분.
- 변수: `lower_with_under`.
- 상수/환경 변수: `UPPER_WITH_UNDER`, 파일 상단에 선언.
- 로컬 변수: 함수 내에서는 항상 `local` 키워드 사용.

기능 및 관례

- Command Substitution: 백틱( ) 대신 `$(command)` 사용.
- Test: `[` 나 `test` 대신 `[[ ... ]]` 사용 권장.
- 산술 연산: `let`이나 `expr` 대신 `(( ... ))` 또는 `$(( ... ))` 사용.
- Wildcards: 파일 이름 확장 시 명시적 경로 사용 (`rm ./*`가 `rm *`보다 안전).
- Error: 모든 오류 메시지는 `STDERR`로 출력.
- Check: `ShellCheck` 도구 사용 강력 권장.