Bash TAB completion

Readline, compspecs, context variables and programmable functions

How completion works

Bash uses the Readline library to edit interactive command lines. Pressing TAB can complete filenames, commands, variables and other arguments. Programmable completion adds command-specific candidates: for example, after systemctl, Bash may suggest units or subcommands rather than arbitrary filenames.

The shell's completion specification (compspec) associates a command with actions such as a shell function, word list or built-in completion generator. The complete builtin registers specifications; compgen produces candidates; compopt adjusts completion behaviour. Installed bash-completion packages often supply hundreds of ready-made definitions.

```
type complete compgen compopt
complete -p | head
shopt -p progcomp
```

The six key variables

COMP_WORDS is the array of words Bash parsed on the command line; COMP_CWORD is the zero-based index of the word containing the cursor. COMPREPLY is the output array your completion function populates. These three form the core of most completion functions.

COMP_LINE holds the whole command line, COMP_POINT is the cursor offset within it, and COMP_WORDBREAKS lists characters treated as separators. Quoting, separators and cursor placement can affect how words are presented. COMP_CWORD is a word index, not a character index. COMPREPLY must be populated with literal candidates, not a formatted message.

```
printf "%s\n" "$COMP_WORDBREAKS"
# Inside a completion function:
local cur=${COMP_WORDS[COMP_CWORD]}
COMPREPLY=( $(compgen -W "start stop status" -- "$cur") )
```

A working completion function

The following function completes the first argument of a fictional command named speak. It uses compgen -W to filter candidates against the current word. In production, consider array-safe methods for filenames containing spaces or unusual characters; command substitution performs word splitting and can lose such values.

The complete -F option registers the function for the command. A completion definition lives in the current shell session unless loaded by startup files or a system completion directory.

```
_speak_complete() {
  local cur=${COMP_WORDS[COMP_CWORD]}
  COMPREPLY=()
  mapfile -t COMPREPLY < <(compgen -W "hello goodbye later extra" -- "$cur")
}
complete -F _speak_complete speak
complete -p speak
```

Argument-sensitive completion

Real commands often have positional and option-specific completion. Examine COMP_CWORD and previous words to determine whether to offer subcommands, options or values. For instance, after a --profile option, suggest profile names rather than subcommands. Keep completion functions fast: they execute during interactive typing, so avoid expensive network calls or unbounded filesystem scans.

Bash can complete filenames using compgen -f and directories using compgen -d. compopt -o filenames asks Readline to quote filenames appropriately. Existing bash-completion helper functions may simplify common patterns, but their availability varies by distribution.

```
_deploy_complete() {
  local cur=${COMP_WORDS[COMP_CWORD]}
  local prev=""
  (( COMP_CWORD > 0 )) && prev=${COMP_WORDS[COMP_CWORD-1]}
  COMPREPLY=()
  if [[ $prev == --profile ]]; then
    mapfile -t COMPREPLY < <(compgen -W "dev test prod" -- "$cur")
  else
    mapfile -t COMPREPLY < <(compgen -W "--profile start stop status" -- "$cur")
  fi
}
complete -F _deploy_complete deploy
```

Installation and debugging

User completion files are often stored under ~/.local/share/bash-completion/completions/ with the filename matching the command; system locations vary. A function sourced by the current shell can be tested immediately. Check that programmable completion is enabled and that another definition has not overridden yours.

If the completion function returns no candidates, inspect COMP_WORDS, COMP_CWORD and the actual COMPREPLY contents. Do not print debug text to standard output during normal completion: it interferes with the interactive line. Use a temporary debug file or Bash tracing while developing.

```
complete -p deploy
complete -r deploy
help complete
help compgen
man bash
```

Reference scope: commands and defaults vary by distribution and software version. Consult the locally installed manual pages for exact syntax and behaviour.