# Neovim 치트시트

## 핵심 명령어 가이드

Neovim은 효율적인 텍스트 편집을 위해 강력하고 다양한 키보드 명령어를 제공합니다. 다음은 자주 사용되는 주요 명령어를 기능별로 분류한 요약입니다.

### 커서 이동 (Navigation)

- `h`: 왼쪽으로 이동
- `j`: 아래로 이동
- `k`: 위로 이동
- `l`: 오른쪽으로 이동
- `0`: 현재 줄의 맨 앞으로 이동
- `^`: 현재 줄에서 공백이 아닌 첫 번째 문자로 이동
- `$`: 현재 줄의 맨 뒤로 이동
- `gg`: 문서의 맨 처음으로 이동
- `G`: 문서의 맨 마지막으로 이동
- `5G`: 5번째 줄로 바로 이동

### 파일 및 버퍼 관리

- `:e <파일명>`: 지정한 파일 열기 및 편집
- `:w`: 현재 버퍼 내용을 저장
- `:w <파일명>`: 현재 내용을 지정한 파일명으로 저장 (다른 이름으로 저장)
- `:q`: 현재 창 닫기
- `:wq`: 저장 후 현재 창 닫기
- `:q!`: 저장하지 않고 강제 종료
- `:wa`: 열려 있는 모든 버퍼 저장

### 편집 기본 (입력 모드 및 기본 조작)

- `i`: 커서 앞에서 입력 시작
- `a`: 커서 뒤에서 입력 시작
- `I`: 현재 줄의 맨 앞에서 입력 시작
- `A`: 현재 줄의 맨 뒤에서 입력 시작
- `o`: 현재 줄 아래에 새 줄을 만들고 입력 시작
- `O`: 현재 줄 위에 새 줄을 만들고 입력 시작

### 고급 편집 및 수정

- `dd`: 현재 줄 전체 삭제
- `D`: 커서 위치부터 줄 끝까지 삭제
- `C`: 커서 위치부터 줄 끝까지 삭제 후 바로 입력 시작
- `u`: 마지막 작업 취소 (Undo)
- `cw`: 커서 위치부터 단어 끝까지 삭제 후 입력 시작
- `cb`: 커서 위치부터 단어 시작까지 삭제 후 입력 시작

### 비주얼 모드 (텍스트 선택)

- `v`: 문자 단위 비주얼 모드 진입
- `V`: 줄 단위 비주얼 모드 진입
- `y`: 선택한 영역 복사 (Yank)
- `d`: 선택한 영역 삭제
- `c`: 선택한 영역 삭제 후 입력 시작

### 검색 및 치환

- `/<패턴>`: 아래 방향으로 특정 패턴 검색
- `?<패턴>`: 위 방향으로 특정 패턴 검색
- `n`: 이전 검색 결과와 같은 방향으로 다음 찾기
- `N`: 이전 검색 결과와 반대 방향으로 다음 찾기
- `:%s/기존/변경/g`: 파일 전체에서 모든 '기존' 패턴을 '변경' 내용으로 치환

### 매크로 및 레지스터

- `qa`: `a` 레지스터에 매크로 기록 시작
- `q`: 매크로 기록 종료
- `@a`: `a` 레지스터에 저장된 매크로 실행
- `"`: 특정 레지스터 지정 시 사용
- `"ay`: 선택 영역을 `a` 레지스터에 복사

### 창 분할 및 탭 관리

- `:sp`: 화면 수평 분할
- `:vsp`: 화면 수직 분할
- `:tabe`: 새 탭 생성
- `:tabc`: 현재 탭 닫기
- `:tabn`: 다음 탭으로 이동
- `:tabp`: 이전 탭으로 이동

## 주요 활용 팁

### 들여쓰기 조절

- `>>`: 현재 줄을 오른쪽으로 들여쓰기
- `<<`: 현재 줄을 왼쪽으로 내어쓰기
- `=`: 코드 스타일 자동 정렬 (현재 줄 또는 선택 영역)
  - `gg=G`: 문서 전체 자동 들여쓰기 정렬

### 복사 및 붙여넣기 관련

- `yy`: 현재 줄 복사 (Yank)
- `dd`: 현재 줄 잘라내기 (Cut)

### 주석 처리 (플러그인 지원 시)

- `gcc`: 현재 줄 주석 토글
- `gc`: 선택 영역 주석 토글

## 실무 단축키 및 테크닉

- `ciw`, `diw`: 현재 단어 내부(inner word) 변경 또는 삭제. `caw`, `daw`는 공백까지 포함.
- `ci(`, `di(`: 괄호 안의 내용만 변경 또는 삭제. `ca(`, `da(`는 괄호 자체 포함.
- `cit`, `dit`: HTML/XML 태그 내부 내용 변경 또는 삭제.
- `:! <명령어>`: 외부 셸 명령어 실행.
- `:r !<명령어>`: 외부 명령어 실행 결과를 현재 커서 위치에 삽입.
- `f<문자>`: 현재 줄에서 특정 문자를 찾아 이동. `;`(다음), `,`(이전)로 반복 검색 가능.
- `Ctrl-a`, `Ctrl-x`: 커서 위치의 숫자 값 증가 또는 감소.

### LSP (Language Server Protocol) 연동

Neovim 내장 LSP를 통해 IDE급 코드 인텔리전스를 활용할 수 있습니다. (설정 필요)

- `gd`: 정의(Definition)로 이동
- `gr`: 참조(References) 목록 확인
- `K`: 현재 심볼 정보 확인 (Hover 정보)
- `[d`, `]d`: 이전/다음 진단(Diagnostic) 결과로 이동
- `<leader>ca`: 코드 액션 실행 (빠른 수정, 리팩토링 제안 등)
- `<leader>rn`: 이름 변경 (Rename)

### 퍼지 파인더 (Telescope, fzf 등) 활용

파일이나 버퍼, 심볼을 빠르게 검색하기 위해 `Telescope` 등 플러그인 사용을 권장합니다.

- 예시 (Telescope 기준):
  - `<leader>ff`: 파일 검색
  - `<leader>fg`: 텍스트 검색 (Live Grep)
  - `<leader>fb`: 현재 열려 있는 버퍼 검색

## 실무 워크플로우 예시

### 코드 리팩토링 과정

```
" 1. 함수 이름 변경
" - 함수 정의로 이동: gd
" - 이름 변경 실행: <leader>rn
" - 참조 위치 확인: gr
" - 모든 참조가 업데이트되었는지 검토

" 2. 코드 추출 및 정리
" - 대상 코드 선택: v (비주얼 모드)
" - 복사: y
" - 함수 추출 액션: <leader>ca (Extract function)
" - 함수 호출 코드로 교체 확인
```

### 디버깅 및 에러 분석

```
" 1. 에러 위치 탐색
" - 진단 목록 창 열기: :lopen
" - 다음 에러로 이동: ]d
" - 에러 상세 정보 확인: K

" 2. 로그 파일 분석
" - 로그 파일 열기: :e /var/log/app.log
" - 특정 패턴 검색: /ERROR
" - 검색 결과 반복: n
" - 필요한 라인 복사: yy
" - 새 창에서 편집: :new (후 p로 붙여넣기)
```

### 파일 관리 및 비교

```
" 1. 프로젝트 탐색
" - 파일 검색: <leader>ff
" - 전역 텍스트 검색: <leader>fg
" - 최근 작업 파일 확인: <leader>fr

" 2. 파일 간 차이점 비교 (Diff)
" - 창 수직 분할: :vsp
" - 비교 대상 파일 열기: :e <파일명>
" - 차이점 강조 시작: :diffthis (양쪽 창에서 실행)
" - 비교 모드 종료: :diffoff
```

## 고급 편집 기법 및 활용

### 매크로 활용을 통한 자동화

```
" 1. 반복 작업 기록 및 실행
" - 기록 시작: qa (a 레지스터 사용)
" - 작업 수행 예시: 줄 끝으로 이동하여 쉼표 추가
"   $a,<Esc>
" - 기록 종료: q
" - 매크로 실행: @a
" - 10번 연속 실행: 10@a

" 2. HTML 태그 감싸기 예시
"   qa
"   I<tag><Esc>
"   A</tag><Esc>
"   j
"   q
" - 실행: @a (현재 줄을 태그로 감싸고 다음 줄로 이동)
```

### 정규표현식 활용 팁

```
" 1. 고급 검색 패턴
" - 모든 함수 정의 찾기: /\vfunction\s+\w+
" - 주석 한꺼번에 제거: :%s/\/\/.*$//g
" - 불필요한 빈 줄 제거: :%s/^\s*$\n//g
```

```
" 2. 복잡한 치환 작업
" - 함수명 일괄 변경(그룹 활용): :%s/\vfunction\s+(\w+)/function new_\1/g
" - 특정 변수만 정확히 매칭하여 변경: :%s/\boldVar\b/newVar/g
```

### 효율적인 버퍼 및 윈도우 관리

```
" 1. 버퍼 운용
" - 목록 확인: :ls 또는 :buffers
" - 특정 버퍼로 전환: :b <버퍼번호>
" - 버퍼 닫기: :bd
" - 모든 변경사항 저장: :wa

" 2. 윈도우 조작
" - 윈도우 간 이동: Ctrl-w + h/j/k/l
" - 윈도우 크기 자동 균등 배분: Ctrl-w =
" - 윈도우 크기 수동 조절: Ctrl-w + / -
```

## 플러그인 기반 워크플로우 확장

### LSP 연동 실전 활용

```
" 1. 탐색 및 정의 확인
" - 정의 이동: gd / 선언 이동: gD
" - 참조 위치 찾기: gr / 구현 위치 찾기: gi

" 2. 리팩토링 및 자동 수정
" - 스마트 이름 변경: <leader>rn
" - 임포트 문 자동 정리: <leader>ca
" - 미사용 코드 제거 제안 실행: <leader>ca
```

### Git 통합 (vim-fugitive 등)

```
" 1. Git 상태 제어
" - 현재 상태 확인: :Git
" - 차이점 비교: :Gdiff
" - 변경 이력 확인: :Git log

" 2. Git 명령어 실행
" - 스테이징: :Gwrite / 커밋: :Git commit
" - 푸시: :Git push / 풀: :Git pull
```

## 성능 및 환경 최적화

### 효율적인 편집을 위한 설정 팁

```
" 1. 매핑 최적화
" - 리더 키 설정: let mapleader = " "
" - 입력 모드 탈출 매핑: inoremap jk <Esc>
" - 빠른 저장 및 종료: nnoremap <leader>w :w<CR>
" 2. 검색 설정 최적화
" - 검색 결과 강조 설정: set hlsearch incsearch
" - 대소문자 스마트 구분: set ignorecase smartcase
```

### 대용량 파일 처리 및 성능 관리

```
" 1. 대용량 파일 전용 모드
" - 플러그인 없이 열기: nvim -u NONE <파일명>
" - 스왑/백업 파일 비활성화: set noswapfile nobackup
" - 언두 기록 비활성화: set noundofile

" 2. 플러그인 로딩 최적화
" - 지연 로딩 활용: packadd! <플러그인명>
" - 특정 파일 타입에서만 로딩 설정
```