Neovim 치트시트

명령어 가이드

Neovim은 효율적인 작업을 위해 다양한 키보드 명령어를 제공합니다. 다음은 자주 사용되는 주요 명령어들을 기능별로 분류하여 정리한 것입니다.

커서 이동 (Moving Around)

- `h`: 커서를 왼쪽으로 이동
- `j`: 커서를 아래로 이동
- `k`: 커서를 위로 이동
- `l`: 커서를 오른쪽으로 이동
- `0`: 커서를 현재 줄의 맨 앞으로 이동
- `^`: 커서를 현재 줄의 첫 번째 비공백 문자로 이동
- `$`: 커서를 현재 줄의 맨 뒤로 이동
- `gg`: 커서를 파일의 맨 앞으로 이동
- `G`: 커서를 파일의 맨 뒤로 이동
- `5G`: 커서를 5번째 줄로 이동

파일 관리 (File Management)

- `:e filename`: 편집할 파일 열기
- `:w`: 현재 버퍼를 디스크에 저장
- `:w filename`: 현재 버퍼를 `filename`으로 저장
- `:q`: 현재 창 닫기
- `:wq`: 저장하고 현재 창 닫기
- `:q!`: 저장하지 않고 현재 창 닫기
- `:wa`: 열려 있는 모든 버퍼를 디스크에 저장

편집 명령 I (Editing Commands I)

- `i`: 커서 앞에 삽입 모드로 진입
- `a`: 커서 뒤에 삽입 모드로 진입
- `I`: 현재 줄의 맨 앞에 삽입 모드로 진입
- `A`: 현재 줄의 맨 뒤에 삽입 모드로 진입
- `o`: 현재 줄 아래에 새 줄을 삽입하고 삽입 모드로 진입
- `O`: 현재 줄 위에 새 줄을 삽입하고 삽입 모드로 진입

편집 명령 II (Editing Commands II)

- `dd`: 현재 줄 삭제
- `D`: 커서부터 줄 끝까지 삭제
- `C`: 커서부터 줄 끝까지 변경하고 삽입 모드로 진입
- `u`: 마지막 변경 사항 실행 취소
- `cw`: 커서부터 단어 끝까지 변경하고 삽입 모드로 진입
- `cb`: 커서부터 단어 시작까지 변경하고 삽입 모드로 진입

비주얼 모드 (Visual Mode Commands)

- `v`: 비주얼 모드 진입 (문자 단위 선택)
- `V`: 비주얼 모드 진입 (줄 단위 선택)
- `y`: 선택된 텍스트 복사 (yank)
- `d`: 선택된 텍스트 삭제
- `c`: 선택된 텍스트 변경하고 삽입 모드로 진입

검색 및 바꾸기 (Search and Replace)

- `/pattern`: `pattern`을 파일에서 앞으로 검색
- `?pattern`: `pattern`을 파일에서 뒤로 검색
- `n`: 이전 검색을 같은 방향으로 반복
- `N`: 이전 검색을 반대 방향으로 반복
- `:%s/pattern/replacement/g`: 파일 전체에서 `pattern`을 `replacement`로 모두 바꾸기

매크로 및 레지스터 (Macros and Registers)

- `qa`: `a` 레지스터에 매크로 기록 시작
- `q`: 매크로 기록 중지
- `@a`: `a` 레지스터에 저장된 매크로 실행
- `"`: 특정 레지스터 접근
- `"ay`: `a` 레지스터에 복사 (yank)

분할 및 탭 (Splits and Tabs)

- `:sp`: 현재 창을 수평으로 분할
- `:vsp`: 현재 창을 수직으로 분할
- `:tabe`: 새 탭 생성
- `:tabc`: 현재 탭 닫기
- `:tabn`: 다음 탭으로 이동
- `:tabp`: 이전 탭으로 이동

유용한 팁

들여쓰기

- `>>`: 현재 줄을 오른쪽으로 들여쓰기
- `<<`: 현재 줄을 왼쪽으로 들여쓰기
- `=`: 자동 들여쓰기 (현재 줄 또는 선택된 영역)
  - `gg=G`: 파일 전체 자동 들여쓰기

복사 및 붙여넣기

- `yy`: 현재 줄 복사 (yank)
- `dd`: 현재 줄 잘라내기 (cut)

주석 처리

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

실무 단축키 및 팁

- `ciw`, `diw`: 현재 단어(inner word)를 변경/삭제. `caw`, `daw`는 공백 포함(a word).
- `ci(`, `di(`: 괄호 안의(inner) 내용을 변경/삭제. `ca(`, `da(`는 괄호 포함.
- `cit`, `dit`: 태그 안의(inner) 내용을 변경/삭제. `cat`, `dat`는 태그 포함.
- `:! <command>`: 셸 명령어 실행.
- `:r !<command>`: 셸 명령어 실행 결과를 현재 커서 위치에 삽입.
- f: 현재 줄에서 를 찾아 앞으로 이동. `;`와 `,`로 반복.
- `Ctrl-a`, `Ctrl-x`: 숫자 증가/감소.

LSP (Language Server Protocol) 연동

Neovim은 내장 LSP 클라이언트를 통해 강력한 코드 인텔리전스 기능을 제공합니다. (LSP 플러그인 설정 필요)

- `gd`: 정의(definition)로 이동.
- `gr`: 참조(references) 목록 보기.
- `K`: 커서 위치의 심볼 정보 보기 (hover).
- `[d`, `]d`: 이전/다음 진단(diagnostic)으로 이동.
- ca: 코드 액션(code action) 실행. (예: 빠른 수정, 리팩토링)
- rn: 이름 변경(rename).

Fuzzy Finder (Telescope, fzf 등)

- 파일, 버퍼, 코드 심볼 등을 빠르게 검색하고 이동하기 위해 `Telescope`나 `fzf.vim` 같은 플러그인 사용을 권장합니다.
- 예시 (Telescope):
  - ff : 파일 검색.
  - fg : 라이브그렙(live grep).
  - fb : 버퍼 검색.

실용적인 워크플로우 예시

코드 리팩토링 워크플로우

```
" 1. 함수명 변경하기
" - 함수 정의로 이동: gd
" - 함수명 변경: <leader>rn
" - 참조 확인: gr
" - 모든 참조 업데이트 확인

" 2. 중복 코드 제거
" - 중복 코드 선택: v (비주얼 모드)
" - 복사: y
" - 함수로 추출: <leader>ca (Extract function)
" - 함수 호출로 교체: <leader>ca (Replace with function call)
```

디버깅 워크플로우

```
" 1. 에러 찾기
" - 진단 확인: :lopen (진단 목록 열기)
" - 다음 에러로 이동: ]d
" - 이전 에러로 이동: [d
" - 에러 정보 보기: K

" 2. 로그 분석
" - 로그 파일 열기: :e /var/log/app.log
" - 에러 패턴 검색: /ERROR
" - 다음 에러: n
" - 에러 라인 복사: yy
" - 새 버퍼에 붙여넣기: :new
```

파일 관리 워크플로우

```
" 1. 프로젝트 탐색
" - 파일 검색: <leader>ff
" - 라이브 그렙: <leader>fg
" - 버퍼 목록: <leader>fb
" - 최근 파일: <leader>fr

" 2. 파일 비교
" - 수직 분할: :vsp
" - 다른 파일 열기: :e filename
" - 차이점 하이라이트: :diffthis
" - 차이점 모드 종료: :diffoff
```

텍스트 편집 마스터 워크플로우

```
" 1. 빠른 편집 패턴
" - 단어 변경: ciw (change inner word)
" - 괄호 내용 변경: ci(, ci[, ci{
" - 따옴표 내용 변경: ci", ci', ci`
" - 태그 내용 변경: cit

" 2. 선택적 편집
" - 현재 단어 선택: viw
" - 현재 문장 선택: vis
" - 현재 문단 선택: vip
" - 현재 함수 선택: vif

" 3. 다중 커서 편집 (플러그인 필요)
" - 같은 단어 선택: <C-n>
" - 다음 단어 추가: <C-n>
" - 편집: i 또는 a
" - ESC로 종료
```

고급 편집 기법

매크로 활용법

```
" 1. 반복 작업 자동화
" - 매크로 기록 시작: qa
" - 작업 수행: (예: 줄 끝으로 이동 후 콤마 추가)
"   $a,<Esc>
" - 매크로 기록 종료: q
" - 매크로 실행: @a
" - 10번 반복: 10@a

" 2. 복잡한 매크로 예시
" - HTML 태그 추가 매크로
"   qa
"   I<tag><Esc>
"   A</tag><Esc>
"   j
"   q
" - 실행: @a (다음 줄에 태그 추가)
```

정규표현식 활용

```
" 1. 고급 검색 및 치환
" - 모든 함수명 찾기: /\vfunction\s+\w+
" - 이메일 주소 찾기: /\v\w+@\w+\.\w+
" - 주석 제거: :%s/\/\/.*$//g
" - 빈 줄 제거: :%s/^\s*$\n//g

" 2. 복잡한 치환 패턴
" - 함수명 변경: :%s/\vfunction\s+(\w+)/function new_\1/g
" - 변수명 일괄 변경: :%s/\boldVar\b/newVar/g
" - 들여쓰기 정리: :%s/^\s\+/\t/g
```

버퍼 및 윈도우 관리

```
" 1. 효율적인 버퍼 관리
" - 버퍼 목록: :ls 또는 :buffers
" - 버퍼 전환: :b <buffer_number>
" - 버퍼 삭제: :bd
" - 모든 버퍼 저장: :wa
" - 모든 버퍼 닫기: :qa

" 2. 윈도우 분할 활용
" - 수평 분할: :sp
" - 수직 분할: :vsp
" - 윈도우 이동: Ctrl-w + h/j/k/l
" - 윈도우 크기 조절: Ctrl-w + +/-
" - 윈도우 균등 분할: Ctrl-w =
```

플러그인 기반 워크플로우

LSP (Language Server Protocol) 활용

```
" 1. 코드 네비게이션
" - 정의로 이동: gd
" - 선언로 이동: gD
" - 참조 찾기: gr
" - 구현 찾기: gi
" - 타입 정의: gT

" 2. 코드 액션 및 리팩토링
" - 코드 액션: <leader>ca
" - 이름 변경: <leader>rn
" - 함수 추출: <leader>ca (Extract function)
" - 임포트 정리: <leader>ca (Organize imports)
" - 불필요한 코드 제거: <leader>ca (Remove unused)

" 3. 진단 및 수정
" - 진단 목록: :lopen
" - 다음 진단: ]d
" - 이전 진단: [d
" - 자동 수정: <leader>ca (Quick fix)
```

Git 통합 (vim-fugitive)

```
" 1. Git 상태 확인
" - Git 상태: :Git
" - 변경사항 확인: :Gdiff
" - 커밋 히스토리: :Git log

" 2. Git 작업
" - 파일 스테이징: :Gwrite
" - 커밋: :Git commit
" - 푸시: :Git push
" - 풀: :Git pull
" - 브랜치 전환: :Git checkout <branch>
```

파일 탐색 (Telescope)

```
" 1. 파일 검색
" - 파일 찾기: <leader>ff
" - 최근 파일: <leader>fr
" - 버퍼 목록: <leader>fb
" - 북마크: <leader>fm

" 2. 코드 검색
" - 라이브 그렙: <leader>fg
" - 심볼 검색: <leader>fs
" - 태그 검색: <leader>ft
" - 명령어 히스토리: <leader>fc
```

성능 최적화 팁

빠른 편집을 위한 설정

```
" 1. 키 매핑 최적화
" - 리더 키 설정: let mapleader = " "
" - ESC 대신 jk 사용: inoremap jk <Esc>
" - 빠른 저장: nnoremap <leader>w :w<CR>
" - 빠른 종료: nnoremap <leader>q :q<CR>

" 2. 검색 최적화
" - 하이라이트 끄기: :nohl
" - 검색 설정: set hlsearch incsearch
" - 대소문자 무시: set ignorecase
" - 스마트 케이스: set smartcase
```

메모리 및 성능 관리

```
" 1. 대용량 파일 처리
" - 큰 파일 열기: vim -u NONE large_file.txt
" - 스왑 파일 비활성화: set noswapfile
" - 백업 비활성화: set nobackup
" - 언두 트리 비활성화: set noundofile

" 2. 플러그인 최적화
" - 지연 로딩: packadd! <plugin>
" - 조건부 로딩: if has('nvim') | packadd! <plugin> | endif
" - 필요시에만 로딩: autocmd FileType python packadd! <plugin>
```