VIM Book

내용

[1 Basic Editing 3](#_Toc178323753)

[시작하기 전에 3](#_Toc178323754)

[Running Vim for the First Time 3](#_Toc178323755)

[Modes 3](#_Toc178323756)

[Editing for the First Time 4](#_Toc178323757)

[Moving Around 4](#_Toc178323758)

[Deleting Characters 4](#_Toc178323759)

[Undo and Redo 4](#_Toc178323760)

[Getting Out 4](#_Toc178323761)

[Discarding Changes 4](#_Toc178323762)

[Other Editing Commands 4](#_Toc178323763)

[Inserting Characters at the End of a Line 4](#_Toc178323764)

[Deleting a Line 4](#_Toc178323765)

[Opening Up New Lines 4](#_Toc178323766)

[Help 5](#_Toc178323767)

[Other Ways to Get Help 7](#_Toc178323768)

[Using a Count to Edit Faster 7](#_Toc178323769)

[The Vim Tutorial 7](#_Toc178323770)

[2 Editing a Little Faster 7](#_Toc178323771)

[Word Movement 7](#_Toc178323772)

[Moving to the Start or End of a Line 8](#_Toc178323773)

[Searching Along a Single Line 8](#_Toc178323774)

[Moving to a Specific Line 8](#_Toc178323775)

[Telling Where You Are in a File 9](#_Toc178323776)

[Where Am I? 9](#_Toc178323777)

[Scrolling Up and Down 10](#_Toc178323778)

[Deleting Text 10](#_Toc178323779)

[Changing Text 10](#_Toc178323780)

[The . Command 10](#_Toc178323781)

[Replacing Characters 11](#_Toc178323782)

[Keyboard Macros 11](#_Toc178323783)

[Digraphs ??? 12](#_Toc178323784)

[3 Searching 13](#_Toc178323785)

[Simple Searches 13](#_Toc178323786)

[Search History 13](#_Toc178323787)

[Searching Options 14](#_Toc178323788)

[Highlighting 14](#_Toc178323789)

[Incremental Searches 14](#_Toc178323790)

[Searching Backward 15](#_Toc178323791)

[Changing Direction 15](#_Toc178323792)

[Basic Regular Expressions 15](#_Toc178323793)

[The Beginning (^) and End ($) of a Line (/와 함께 사용) 15](#_Toc178323794)

[Match Any Single Character (.) (/와 함께 사용) 16](#_Toc178323795)

[Matching Special Characters 16](#_Toc178323796)

[4 Text Blocks and Multiple Files 16](#_Toc178323797)

[Cut, Paste, and Copy 16](#_Toc178323798)

[A. Installing Vim 17](#_Toc178323799)

[UNIX 17](#_Toc178323800)

# 1 Basic Editing

이 장에서는 편집을 시작하는 데 필요한 10가지 기본 Vim 명령을 알려줍니다. 이 장에서는 다음을 배웁니다.

* 네 가지 기본 이동 명령
* 텍스트를 삽입하고 삭제하는 방법
* 도움말을 얻는 방법(매우 중요)
* 편집기 종료

## 시작하기 전에

Vim을 설치하지 않은 경우 부록 A, "Vim 설치"를 읽고 편집기를 설치해야 합니다.

UNIX에서 실행 중이라면 다음 명령을 실행하세요.

$ touch ~/.vimrc  
~/.vimrc를 생성하면 Vim 모드에서 사용하고 싶다고 Vim에 알립니다. 이 파일이 없으면 Vim은 Vi 호환 모드에서 실행되고 많은 고급 Vim 기능에 액세스할 수 없게 됩니다. 그러나 언제든지 다음 명령을 사용하여 Vim 내에서 고급 기능을 활성화할 수 있습니다. :set nocompatible<Enter>.

## Running Vim for the First Time

Vim을 시작하려면 다음 명령을 입력하세요.

$ gvim file.txt

### Modes

Vim 편집기는 모달 편집기입니다. 즉, 편집기는 모드에 따라 다르게 동작합니다. 화면 하단에 파일 이름이 표시되거나 비어 있으면 일반 모드입니다. 삽입 모드에 있는 경우 표시기에 --INSERT--가 표시되고, 시각적 모드에 있는 경우 표시기에 --VISUAL--이 표시됩니다.

## Editing for the First Time

### Moving Around

h, j, k, l

### Deleting Characters

문자를 삭제하려면 커서를 해당 문자 위로 옮기고 x를 입력합니다. (이것은 옛날 타자기 시절로 돌아가서, xxxx를 입력해서 삭제하던 방식입니다.)

### Undo and Redo

너무 많은 내용을 삭제했다고 가정해 보겠습니다. 다시 입력할 수도 있지만 더 쉬운 방법이 있습니다. **u** 명령은 마지막 편집 내용을 취소합니다.

너무 많이 실행 취소하면 **CTRL-R**(다시 실행)을 눌러 이전 명령을 되돌릴 수 있습니다. 즉, 실행 취소를 취소합니다.

### Getting Out

종료하려면 **ZZ** 명령을 사용합니다. 이 명령은 파일을 쓰고 종료합니다.

### Discarding Changes

**:q!**

자세한 내용에 관심이 있는 분들을 위해, 이 명령의 세 부분은 콜론(:)으로, 명령 모드로 들어갑니다. q 명령은 편집기를 종료하라고 알려줍니다. override 명령 수정자(!)입니다.

## Other Editing Commands

### Inserting Characters at the End of a Line

**i** 명령은 커서 아래의 문자 앞에 문자를 삽입합니다.

커서 뒤에 텍스트를 삽입하는 작업은 **a** (append) 명령으로 수행합니다.

### Deleting a Line

줄을 삭제하려면 **dd** 명령을 사용합니다.

### Opening Up New Lines

새로운 줄을 추가하려면 **o** 명령을 사용하여 커서 아래에 새 줄을 엽니다.그러면 편집기가 삽입 모드로 전환됩니다.

## Help

마지막으로, 또 하나의 중요한 명령이 있습니다. 바로 help 명령입니다. 도움말을 얻으려면 다음을 입력하세요.

**:help**

주제를 제공하지 않으면 :help는 일반 도움말 창을 표시합니다. Vim의 제작자는 도움말 시스템에 매우 영리한(또는 매우 게으른) 일을 했습니다. 그들은 help 창을 일반 편집 창으로 만들었습니다. 모든 일반 Vim 명령을 사용하여 도움말 정보를 이동할 수 있습니다. 따라서 h, k, j, l은 왼쪽, 위, 아래, 오른쪽 등으로 이동합니다.

도움말 시스템을 종료하려면 편집기를 종료하는 데 사용하는 것과 동일한 명령인 ZZ를 사용하세요.

도움말 텍스트를 읽으면 세로 막대로 둘러싸인 텍스트가 보일 것입니다(예: |:help|). 이는 하이퍼링크를 나타냅니다. 커서를 막대 사이 아무 데나 놓고 CTRL+](태그로 이동)를 누르면 도움말 시스템이 지정된 주제로 이동합니다. (여기서는 설명하지 않은 이유로, 하이퍼링크에 대한 Vim 용어는 태그입니다. 따라서 CTRL+]는 커서 아래에 있는 단어가 지정한 태그 위치로 이동합니다.)

몇 번 점프한 후에는 돌아가고 싶을 수도 있습니다. CTRL+T(태그 팝)는 이전 화면으로 돌아갑니다. 또는 Vim 용어로 "태그 스택에서 태그를 팝합니다."

이 화면의 맨 위에는 \*help.txt\*라는 표기가 있습니다. 이것은 도움말 시스템에서 태그(하이퍼링크 대상)를 정의하는 데 사용됩니다. 7장, “프로그래머를 위한 명령”에서 태그에 대해 자세히 설명합니다.

주어진 주제에 대한 도움말을 얻으려면 다음 명령을 사용하세요.

**:help subject**

예를 들어, x 명령에 대한 도움말을 보려면 다음을 입력하세요.

:help x

텍스트를 삭제하는 방법을 알아보려면 다음 명령을 사용하세요.

:help deleting

사용 가능한 전체 인덱스를 보려면 다음 명령을 사용하세요.

:help index

제어 문자 명령(예: CTRL-A)에 대한 도움말이 필요할 때는 접두사 CTRL-를 붙여 입력해야 합니다.

:help CTRL-A

Vim 편집기에는 다양한 모드가 있습니다. 기본적으로 도움말 시스템은 일반 모드 명령을 표시합니다. 예를 들어, 다음 명령은 일반 모드 CTRL-H 명령에 대한 도움말을 표시합니다.

:help CTRL-H

다른 모드를 식별하려면 모드 접두사를 사용합니다.

이 명령의 삽입 모드 버전에 대한 도움말이 필요하면 키 앞에 i\_를 붙입니다. 그러면 다음 명령이 생성됩니다.

:help i\_CTRL-H

표 1.1에는 여러 가지 다른 모드 접두사가 나열되어 있습니다.

Vim 편집기를 시작하면 여러 명령줄 옵션을 사용할 수 있습니다. 이러한 옵션은 모두 대시(-)로 시작합니다. 예를 들어 -t 명령줄 옵션이 무엇을 하는지 알아보려면 명령을 사용합니다.

:help -t

Vim 편집기에는 편집기를 구성하고 사용자 정의할 수 있는 여러 옵션이 있습니다. 옵션에 대한 도움말이 필요한 경우 작은 따옴표로 묶어야 합니다. 예를 들어 숫자 옵션이 무엇을 하는지 알아보려면 다음 명령을 사용합니다.

:help ‘number’

다음 표는 특수 접두사를 요약한 것입니다.

텍스트, 폰트, 번호, 라인이(가) 표시된 사진

자동 생성된 설명

특수 키는 꺾쇠 괄호로 묶습니다. 예를 들어 위쪽 화살표 키에 대한 도움말을 찾으려면 다음 명령을 사용합니다.

:help <Up>

부록 B, “<> 키 이름”에서는 키 이름의 전체 목록을 제공합니다.

### Other Ways to Get Help

<F1> 키를 눌러 도움말 화면으로 이동할 수 있습니다. 그러면 일반 도움말 화면이 표시되고, 거기에서 탐색할 수 있습니다. 키보드에 <Help> 키가 있으면 사용할 수도 있습니다.

## Using a Count to Edit Faster

예를 들어, 9줄 위로 이동하고 싶다면 kkkkkkkkk를 입력하거나 **9k** 명령을 입력하면 됩니다.

사실, 모든 이동 명령 앞에 숫자를 붙일 수 있습니다. 예를 들어, 이 장의 앞부분에서 a!!!<Esc>를 입력하여 줄 끝에 느낌표 세 개를 추가했습니다. 이를 수행하는 또 다른 방법은 **3a!<Esc>** 명령을 사용하는 것입니다. 3의 카운트는 a 명령에 뒤에 오는 것을 (!) 세 번 삽입하라고 말합니다.

마찬가지로, 세 개의 문자를 삭제하려면 **3x** 명령을 사용합니다.

## The Vim Tutorial

Vim 편집기의 UNIX 버전에는 대화형 튜토리얼이 제공됩니다. 1과에서는 이 장에서 설명하는 많은 명령을 다룹니다.

UNIX에서 튜토리얼을 호출하려면 다음 명령을 사용합니다.

**$ vimtutor**

# 2 Editing a Little Faster

## Word Movement

이동부터 시작해 보겠습니다. 커서를 한 단어 앞으로 이동하려면 **w** 명령을 사용합니다. **b** 명령은 한 단어 뒤로 이동합니다. 대부분의 Vim 명령과 마찬가지로 숫자 접두사를 사용하여 여러 단어를 지나 이동할 수 있습니다. 예를 들어, **4b**는 네 단어 뒤로 이동합니다.

## Moving to the Start or End of a Line

**$** 명령은 커서를 줄의 끝으로 옮깁니다. 사실, 여러 키가 "줄 끝" 명령에 매핑됩니다. 이러한 키에 대한 Vim 이름은 $, **<End>**, <kEnd>입니다. (<kEnd> 키는 Vim에서 키패드 End 키에 대한 이름입니다.)

$ 명령은 숫자 인수도 받습니다. 존재하는 경우 편집기가 다음 줄의 끝으로 이동합니다. 예를 들어, **1$**는 첫 번째 줄(현재 줄)의 끝으로 이동하고, **2$**는 다음 줄의 끝으로 이동합니다.

**^** 명령은 줄의 첫 번째 공백이 아닌 문자로 이동합니다. **<Home>** 또는 <Home> 키는 그림 2.3에서 볼 수 있듯이 줄의 첫 번째 문자로 이동합니다. (**0** [영] 명령도 동일한 작업을 수행합니다.)

## Searching Along a Single Line

이동은 가장 일반적인 편집 활동입니다. 가장 유용한 이동 명령 중 하나는 단일 문자 검색 명령입니다. 명령 **fx(전방 검색)**는 줄에서 단일 문자 x를 검색합니다.  
예를 들어, 다음 줄의 시작 부분에 있다고 가정해 보겠습니다.  
To err is human. To really foul up you need a computer.  
human의 h로 이동하고 싶다고 가정해 보겠습니다. **fh** 명령을 실행하면 커서가 h 위에 위치합니다.  
To err is human. To really foul up you need a computer.  
**5f<Space>**: 명령을 사용하여 5개 단어를 앞으로 스페이스할 수 있습니다.  
To err is human. To really foul up you need a computer.

**F** 명령은 왼쪽으로 검색합니다. 그림 2.4는 f 및 F 명령의 효과를 보여줍니다.

**tx(search ‘til)** 명령은 fx 명령과 비슷하게 작동하지만 지정된 문자 바로 앞에서 한 문자만 멈춥니다. 이 명령의 이전 버전은 **Tx**입니다.

때로는 검색을 시작하다가 잘못된 명령을 입력한 것을 깨닫는 경우가 있습니다. 예를 들어, 뒤로 검색하려면 f를 입력하지만 실제로는 F를 의미했다는 것을 깨닫습니다. 검색을 중단하려면 검색 키로 **<Esc>**를 누릅니다. 따라서 f<Esc>는 뒤로 검색을 중단한 것입니다. (참고: <Esc>는 검색뿐만 아니라 대부분의 작업을 취소합니다.)

## Moving to a Specific Line

C 또는 C++ 프로그래머라면 다음과 같은 오류 메시지를 알고 있을 것입니다.

prog.c:3: 'j'가 선언되지 않음(이 함수에서 처음 사용됨)

이것은 3번째 줄에서 무언가를 수정하고 싶을 수도 있다는 것을 알려줍니다. 그러면 3번째 줄을 어떻게 찾을까요?

한 가지 방법은 9999k를 사용하여 파일의 맨 위로 이동하고 2j를 사용하여 두 줄 아래로 이동하는 것입니다. 좋은 방법은 아니지만 작동합니다.

훨씬 더 나은 방법은 **G** 명령을 사용하는 것입니다. 인수를 사용하면 이 명령은 지정된 줄 번호로 이동합니다. 예를 들어 **3G**는 3번째 줄로 이동합니다. (마찬가지로 9999k가 아닌 **1G** 명령을 사용하여 파일의 맨 위로 이동합니다.)

인수가 없으면 파일의 끝에 위치합니다.

(컴파일러 오류 목록을 살펴보는 더 나은 방법은 7장 "프로그래머를 위한 명령"에서 :make 및 :clist 관련 명령에 대한 정보를 참조하세요.)

## Telling Where You Are in a File

파일에서 어디에 있는지 실제로 어떻게 알 수 있을까요? 여러 가지 방법으로 알 수 있습니다. 첫 번째는 다음 명령으로 줄 번호를 켜는 것입니다(그림 2.6 참조):

**:set number**

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

Vim 편집기는 구성 가능성이 매우 높고 옵션이 매우 많습니다. :set 명령은 다양한 방법으로 사용할 수 있으며, 이에 대해서는 28장 "편집기의 모양과 동작 사용자 정의"에서 설명합니다.

끄려면 다음 명령을 사용하세요.

**:set nonumber**

### Where Am I?

**CTRL-G** 명령은 파일에서 어디에 있는지를 나타내는 상태 줄을 표시합니다. 예:

“c02.txt” [Modified] line 81 of 153 —52%— col 1

이것은 c02.txt라는 파일을 편집 중이며 편집이 시작된 이후로 수정되었음을 나타냅니다. 커서는 총 153개 중 81번째 줄, 즉 파일의 약 52%에 위치합니다. 커서는 현재 1열에 있습니다.

## Scrolling Up and Down

**CTRL-U** 명령은 반 화면의 텍스트를 위로 스크롤합니다.

**CTRL-D** 명령은 반 화면 아래로 스크롤합니다.

## Deleting Text

1장에서 배웠듯이 dd 명령은 줄을 삭제합니다. **dw** 명령은 단어를 삭제합니다. w 명령은 move word 명령으로 알고 계실 겁니다. 사실 d 명령은 모든 motion 명령 뒤에 올 수 있으며, 현재 위치에서 커서가 끝나는 곳까지 삭제합니다. (따라서 d 명령의 구문은 dmotion이라고 합니다.)

예를 들어 3w 명령은 커서를 세 단어 위로 이동합니다. **d3w** 명령은 그림 2.10에서 볼 수 있듯이 세 단어를 삭제합니다. (d3w 또는 3dw로 쓸 수 있습니다. 두 버전 모두 동일하게 작동합니다.)

$ 명령은 줄의 끝으로 이동합니다. **d$** 명령은 그림 2.11에서 볼 수 있듯이 커서에서 줄의 끝까지 삭제합니다. 이 명령의 단축키는 **D** 명령입니다.

## Changing Text

**c** 명령은 텍스트를 변경합니다. **d 명령과 똑같이 작동하지만, 삽입 모드로 전환**됩니다. 예를 들어, **cw**는 단어를 변경합니다. 또는 더 구체적으로, 단어를 삭제한 다음 삽입 모드로 전환합니다.

cmotion 명령은 dmotion 명령과 똑같이 작동하지만, cw 및 dw 명령은 예외입니다. cw는 단어 뒤의 공백까지 텍스트를 삭제한 다음 삽입 모드로 전환하는 반면, dw 명령은 단어와 그 뒤의 공백을 삭제합니다.

**cc** 명령은 전체 줄에서 작동합니다. 즉, 줄을 삭제한 다음 삽입 모드로 전환합니다. 즉, cc는 dd와 마찬가지로 현재 줄에서 작동합니다. 마찬가지로 **c$** 또는 **C**는 커서에서 줄 끝까지 변경합니다.

## The . Command

**. 명령**은 Vim에서 가장 간단하면서도 강력한 명령 중 하나입니다. 마지막 삭제 또는 변경 명령을 반복합니다. 예를 들어 HTML 파일을 편집하고 모든 <B> 태그를 삭제하려고 한다고 가정합니다. 커서를 첫 번째 <에 놓고 **df>** 명령으로 <B>를 삭제합니다. 그런 다음 다음 </B>의 <로 이동하여 . 명령을 사용하여 종료합니다. . 명령은 마지막 변경 명령(이 경우 df>)을 실행합니다. 다른 태그를 삭제하려면 커서를 <에 놓고 . 명령을 누릅니다.

**Joining Lines**

**J** 명령은 현재 줄을 다음 줄과 연결합니다. 그림 2.14에서 볼 수 있듯이, 첫 번째 줄의 끝에 공백을 추가하여 연결된 두 부분을 구분합니다.

카운트를 지정하면 카운트 줄이 연결됩니다(최소 두 개). 예: **3J**

## Replacing Characters

**rx** 명령은 커서 아래의 문자를 x로 바꿉니다. 그림 2.15는 r 명령을 사용하여 z를 s로 바꾸는 방법을 보여줍니다.

r 명령 앞에는 바꿀 문자 수를 나타내는 count를 붙일 수 있습니다. 그림 2.16에서 줄의 시작(^ 명령)으로 이동하여 **5ra**를 실행하여 처음 다섯 문자를 a로 바꿉니다.

참고  
r 명령은 <Enter>를 특별한 방식으로 처리합니다. 카운트가 아무리 크더라도 <Enter>는 하나만 삽입됩니다. 따라서 5ra는 다섯 개의 a 문자를 삽입하는 반면 **5r<Enter>**는 다섯 개의 문자를 하나의 <Enter>로 대체합니다.

카운트를 어디에 두는지 주의하세요.5rx 명령은 5개의 문자를 x 문자로 바꾸는 반면, r5x는 커서 아래의 문자를 5(r5)로 바꾼 다음 문자(x)를 삭제합니다.

**Changing Case**

**~** 명령은 문자의 대소문자를 변경합니다. 대문자를 소문자로, 그리고 그 반대로 변경합니다. 개수를 지정하면 개수 문자가 변경됩니다. 예: **14~**

## Keyboard Macros

. 명령은 이전 변경 사항을 반복합니다. 하지만 단일 변경보다 더 복잡한 작업을 하고 싶다면 어떨까요? 바로 키보드 매크로가 등장합니다. **q**character 명령은 키 입력을 character라는 레지스터에 기록합니다. (character는 a와 z 사이여야 합니다.)

기록을 마치려면 **q** 명령을 입력하기만 하면 됩니다. 이제 **@character** 명령을 입력하여 매크로를 실행할 수 있습니다. (이 명령 앞에 카운트를 입력하면 매크로가 해당 횟수만큼 실행됩니다.)

실제로 이러한 명령을 사용하는 방법을 살펴보겠습니다. 다음과 같은 파일 이름 목록이 있습니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

그리고 당신이 원하는 것은 다음과 같습니다:

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

첫 번째 줄의 첫 번째 문자로 이동하면서 시작합니다. 다음으로 다음 명령을 실행합니다.

|  |  |  |
| --- | --- | --- |
|  | qa | 레지스터 a에서 매크로 기록을 시작합니다. |
|  | ^ | 줄의 시작 부분으로 이동합니다. |
|  | i#include “<Esc> | 줄의 시작부분에 #include " 문자열을 삽입합니다. |
|  | $ | 줄의 끝으로 이동합니다. |
|  | a”<Esc> | 줄의 끝에 큰따옴표(")를 추가합니다. |
|  | j | 다음 줄로 이동합니다. |
|  | q | 매크로 기록을 중지합니다. |

이제 한 번 작업을 마쳤으니 **@a** 명령을 입력하여 변경을 반복할 수 있습니다. 또는 세 줄이 남았으므로 **3@a** 명령을 사용하여 변경할 수 있습니다.

## Digraphs ???

일부 문자는 키보드에 없습니다. 예를 들어 저작권 문자(©)가 있습니다.

Vim에서 이러한 문자를 입력하려면 두 문자가 하나를 나타내는 이중 문자를 사용합니다. 예를 들어 ©를 입력하려면 **CTRL-Kc0**을 입력합니다.

사용 가능한 이중 문자를 찾으려면 다음 명령을 사용합니다.

**:digraphs**

Vim 편집기는 그림 2.19에서 볼 수 있듯이 이중 문자 매핑 테이블을 표시합니다.

예를 들어 CTRL-K~!를 입력하여 얻은 이중 문자는 문자(¡)입니다. 이것은 문자 번호 161입니다.

경고

이 이중 문자는 표준 ISO-646 문자 집합이 있다고 가정하여 설정되었습니다. 이는 국제 표준이지만 특정 디스플레이 또는 인쇄 시스템에서는 사용하지 않을 수 있습니다.

# 3 Searching

* Simple forward searches
* Search options
* Incremental searches
* Changing directions
* Basic regular expressions

## Simple Searches

문자열을 검색하려면 **/string** 명령을 사용합니다. 예를 들어 include라는 단어를 찾으려면 /include 명령을 사용합니다. 이 명령의 끝에는 <Enter>가 암시되어 있습니다.

(커서가 화면 하단으로 이동하고 무언가를 입력할 때마다 <Enter>로 끝내야 합니다.)

참고: 문자 .\*[]ˆ%/\?~$는 특별한 의미를 갖습니다. 검색에 사용하려면 문자 앞에 \를 넣어야 합니다. 예: .을 찾으려면 검색 문자열 \..을 사용합니다.

다음 include를 찾으려면 **/<Enter>** 명령을 사용합니다.  
다음 일치 항목을 찾는 또 다른 방법은 **n** 명령을 사용하는 것입니다. 이 명령은 /<Enter>와 동일한 작업을 수행하지만 키 입력이 하나 적습니다.

/<Enter> 및 n 명령은 모두 카운트를 지정할 수 있습니다. 카운트가 있는 경우 명령은 현재 위치에서 카운트 수의 일치 항목을 검색합니다.

### Search History

검색 명령에는 기록 기능이 있습니다. 예를 들어, 세 가지 검색을 한다고 가정해 보겠습니다.  
 /one  
 /two  
 /three

이제 <Enter>를 누르지 않고 간단한 **/**를 입력하여 검색을 시작해 보겠습니다. <Up>을 누르면 Vim은 프롬프트 줄에 /three을 입력합니다. 이 시점에서 <Enter>를 누르면 three를 검색합니다. <Enter>를 누르지 않고 대신 <Up>을 누르면 Vim은 프롬프트를 /two로 변경합니다. 또 다른 <Up> 명령은 /one으로 이동합니다.

즉, 여러 번 검색한 후 **<Up>** 및 **<Down>** 키를 사용하여 최근 검색 중 하나를 선택할 수 있습니다.

## Searching Options

검색을 수행하는 방법을 제어하는 ​​데는 다양한 옵션이 있습니다. 이 섹션에서는 그 중 몇 가지에 대해 설명합니다.

### Highlighting

다음 명령을 사용하면 Vim에서 검색 패턴과 일치하는 문자열을 강조 표시합니다.

**:set hlsearch**

이 옵션을 켜고 예를 들어 include를 검색하면 모든 include 문자열의 결과가 강조 표시됩니다(그림 3.4 참조).

검색 강조 표시를 끄려면 다음 명령을 사용하세요.

**:set nohlsearch**

현재 강조 표시를 지우려면 다음 명령을 사용하세요.

**:nohlsearch**

검색 강조 표시가 꺼졌습니다. 일치하는 텍스트는 강조 표시되지 않습니다. 그러나 검색 명령을 사용하면 강조 표시가 다시 표시됩니다.

### Incremental Searches

기본적으로 Vim은 기존 검색 방법을 사용합니다. 문자열을 지정하면 Vim이 검색을 수행합니다. 다음 명령을 사용하면 편집기가 증분 검색을 수행합니다.

**:set incsearch**

문자열의 첫 번째 문자를 입력하자마자 편집기가 검색을 시작합니다. 각 추가 문자는 검색을 더욱 세분화합니다.

예를 들어, ioctl.h를 검색하고 싶지만 이번에는 증분 검색을 사용하고 싶다고 가정해 보겠습니다. 먼저 증분 검색을 켭니다.

다음으로, /i 명령을 입력하여 검색을 시작합니다. 첫 번째 i를 검색하고 커서를 그 위에 놓는 방법을 보여줍니다.

o를 입력하여 검색을 계속합니다. 이제 검색어는 /io이므로 편집기는 첫 번째 io를 찾습니다.

여전히 원하는 위치는 아니므로 검색에 c를 추가하여 /ioc 명령을 생성합니다. Vim 편집기는 ioc와 일치하는 첫 번째 위치로 이동합니다.

이것이 찾으려는 것이므로 <Enter>를 누르면 됩니다.

증분 검색을 끄려면 다음 명령을 사용합니다.

**:set noincsearch**

## Searching Backward

역방향 검색 명령(**?**)은 뒤로 검색합니다. n 명령은 마지막 검색을 반복합니다. 역방향 검색이 마지막으로 사용된 경우 n 명령은 역방향으로 검색합니다. 마지막 검색이 정방향 검색인 경우 n 명령은 정방향으로 검색합니다.

## Changing Direction

/unsigned 명령을 사용하여 unsigned에 대한 전방 검색을 시작한다고 가정해 보겠습니다. 명령을 사용하여 방향을 바꾸어 역방향으로 검색할 수 있습니다. n 명령은 같은 방향으로 검색을 반복합니다. **N** 명령은 검색 방향을 반대로 바꾸어 반복합니다.

## Basic Regular Expressions

Vim 편집기는 정규 표현식을 사용하여 검색할 내용을 지정합니다. 정규 표현식은 검색 패턴을 지정하는 매우 강력하고 간결한 방법입니다. 안타깝게도 이 힘은 정규 표현식을 지정하기가 약간 까다로워서 비용이 많이 듭니다.

간단한 것부터 시작해 보겠습니다. 정규 표현식에서 일반 문자는 스스로 매치됩니다. 따라서 정규 표현식 Steve는 Steve와 매치됩니다.

### The Beginning (^) and End ($) of a Line (/와 함께 사용)

^ 문자는 줄의 시작과 일치합니다. (이것이 줄의 시작으로 이동하는 명령인 것은 우연이 아닙니다.) 표현식 include는 줄의 어느 곳에서나 include라는 단어와 일치합니다. 하지만 표현식 ^include는 줄의 시작에 있을 때만 include라는 단어와 일치합니다.

$ 문자는 줄의 끝과 일치합니다. 따라서 **was$**는 줄의 끝에 있는 경우에만 was라는 단어를 찾습니다. 예를 들어 그림 3.10은 강조 표시를 활성화한 패턴 the에 대한 검색을 보여줍니다.

다음으로 정규 표현식 **^the**를 검색할 때 어떤 일이 발생하는지 살펴보겠습니다. 그림 3.11에서 볼 수 있듯이 결과는 줄을 시작하는 두 개의 항목만 강조 표시되어 있음을 보여줍니다.

마지막으로 the$를 검색합니다. 그림 3.12에서 볼 수 있듯이 줄을 끝내는 것은 the 중 하나뿐입니다. the라는 단어로만 구성된 줄을 검색하려면 정규 표현식 ^the$를 사용합니다. 빈 줄을 검색하려면 정규 표현식 ^$를 사용합니다.

### Match Any Single Character (.) (/와 함께 사용)

문자 .는 모든 단일 문자와 일치합니다. 예를 들어, 표현식 **c.m**은 첫 번째 문자가 c이고, 두 번째 문자가 무엇이든, 세 번째 문자가 m인 문자열과 일치합니다. 그림 3.13은 패턴이 computer의 com과 became의 cam과 일치함을 보여줍니다.

### Matching Special Characters

대부분의 기호는 정규 표현식 내부에서 특별한 의미를 갖습니다. 이러한 특수 기호를 일치시키려면 기호 앞에 백슬래시(\)를 붙여야 합니다. 예를 들어, 마침표(.)를 찾으려면 문자열 the\..를 사용합니다.

# 4 Text Blocks and Multiple Files

대부분 편집기에서는 잘라내기와 붙여넣기만 하면 됩니다. 그러나 Vim 편집기에는 레지스터라는 개념이 있습니다. 이를 통해 여러 잘라내기, 복사 또는 붙여넣기 작업에 대한 데이터를 보관할 수 있습니다. 대부분의 다른 편집기는 단일 잘라내기/붙여넣기 클립보드로 제한됩니다. Vim 레지스터를 사용하면 26개 이상의 클립보드를 얻을 수 있습니다.

UNIX의 장점 중 하나는 다양한 텍스트 조작 명령을 제공한다는 것입니다. 이 장에서는 filter 명령을 사용하여 UNIX 필터를 사용하여 Vim에서 텍스트를 편집하는 방법을 보여줍니다.

지금까지 이 책에서는 단일 파일을 사용하여 작업했습니다.이제 여러 파일을 사용하게 될 것입니다.이렇게 하면 여러 파일에서 동일한 편집 작업을 수행하고, 파일 간에 잘라내기 및 붙여넣기 작업이 가능해집니다.

* 간단한 잘라내기 및 붙여넣기 작업(Vim 용어로 delete 및 put)
* 텍스트 내 위치 표시
* yank 명령을 사용하여 텍스트를 레지스터에 복사
* 텍스트 필터링
* 여러 파일 편집

## Cut, Paste, and Copy

d, x 또는 다른 명령으로 무언가를 삭제하면 텍스트가 저장됩니다. p 명령을 사용하여 다시 붙여넣을 수 있습니다. (이것의 기술적 이름은 put입니다).

먼저 dd 명령으로 전체 줄을 삭제합니다. 삭제하려는 줄에 커서를 놓고 dd를 누릅니다. 이제 줄을 놓을 곳으로 커서를 옮기고 **p(put)** 명령을 사용합니다. 줄은 커서 다음 줄에 삽입됩니다.

줄 전체를 삭제했기 때문에 p 명령은 커서 뒤의 줄에 텍스트를 배치했습니다.

줄의 일부(예를 들어 dw 명령으로 단어)를 삭제하는 경우 p 명령은 커서 아래 문자 바로 뒤에 텍스트를 배치합니다

### Character Twiddling

타이핑을 할 때 손가락이 뇌보다 앞서는 경우가 많습니다. 그 결과 teh와 같은 오타가 생깁니다. Vim 편집기를 사용하면 이러한 문제를 쉽게 수정할 수 있습니다. teh의 e에 커서를 놓고 **xp** 명령을 실행하기만 하면 됩니다.

### More on “Putting”

**p** 명령을 여러 번 실행할 수 있습니다. 매번 텍스트의 다른 사본을 파일에 삽입합니다.

p 명령은 텍스트를 커서 뒤에 놓습니다. **P** 명령은 텍스트를 커서 앞에 놓습니다. 두 명령 모두에 count를 사용할 수 있으며, 지정된 경우 텍스트가 count 번 삽입됩니다. 예: 4p

## Marks

Vim 편집기를 사용하면 텍스트에 마크를 넣을 수 있습니다. **ma** 명령은 커서 아래의 위치를 ​​마크 a로 표시합니다. 텍스트에 26개 마크(a에서 z까지)를 넣을 수 있습니다. (다른 마크도 여러 개 사용할 수 있습니다.)

마크로 이동하려면 **`mark`** 명령을 사용합니다. 여기서 mark는 마크 문자이고 `는 백틱 문자나 작은따옴표를 여는 문자입니다.

'mark(작은따옴표 또는 아포스트로피)' 명령을 사용하면 해당 마크가 포함된 줄의 시작 부분으로 이동합니다. 이 명령은 표시된 줄과 열로 이동하는 `mark' 명령과 다릅니다.

'mark' 명령은 긴 줄을 삭제할 때 매우 유용할 수 있습니다. 긴 줄을 삭제하려면 다음 단계를 따르세요.

1. 커서를 삭제하려는 텍스트의 시작 부분으로 이동합니다.
2. ma 명령을 사용하여 표시합니다. (이 명령은 a로 표시합니다.)
3. 삭제할 텍스트의 끝으로 이동합니다. d’a 명령을 사용하여 a로 표시합니다.

참고: a 표시를 사용하는 데 특별한 것은 없습니다. a에서 z까지 어떤 표시든 사용할 수 있습니다.

먼저 시작 부분을 작업한 다음 끝 부분을 작업하는 데는 특별한 것이 없습니다. 끝 부분을 표시한 다음 커서를 시작 부분으로 옮긴 후 표시한 부분까지 삭제해도 됩니다.

마크의 장점 중 하나는 텍스트가 이동하더라도(마크 위에 텍스트를 삽입하거나 삭제한 경우) 텍스트와 함께 유지된다는 것입니다. 물론, 마크가 포함된 텍스트를 삭제하면 마크가 사라집니다.

### Where Are the Marks?

모든 mark를 나열하려면 다음 명령을 사용하세요.

**:marks**

## Yanking

d 명령을 사용하여 삭제하고, p 명령으로 삭제된 텍스트를 복원한 다음, 복사본을 원하는 곳으로 가서 p를 사용하여 텍스트에 넣었습니다.

더 나은 방법이 있습니다. **y** 명령은 텍스트를 레지스터에 "yank"합니다(파일에서 제거하지 않고). y 명령의 일반 형식은 ymotion입니다. delete(d) 명령과 똑같이 작동하지만 텍스트는 삭제되지 않습니다. 그리고 단축형 **yy**는 현재 줄을 버퍼에 yank합니다.

먼저 복사할 텍스트의 맨 위로 가서 ma로 표시합니다. 그런 다음 맨 아래로 가서 **y’a**(yank하여 a 표시)를 합니다.

이제 복사한 텍스트를 삽입할 위치로 가서 p 명령을 사용하여 거기에 넣습니다.

## Filtering

**!**motion 명령은 텍스트 블록을 가져와 다른 프로그램으로 필터링합니다. 즉, command로 표현된 시스템 명령을 실행하여 motion으로 표현된 텍스트 블록을 입력으로 제공합니다. 그런 다음 이 명령의 출력이 선택된 블록을 대체합니다.

UNIX 필터에 익숙하지 않다면 요약이 잘 안 될 수 있으므로 예를 살펴보세요. sort 명령은 파일을 정렬합니다. 다음 명령을 실행하면 정렬되지 않은 파일 input.txt가 정렬되어 output.txt에 기록됩니다.

$ sort <input.txt >output.txt

이제 Vim에서 같은 작업을 해보세요. 파일의 1~10번째 줄을 정렬하고 싶습니다. 커서를 1번째 줄에 두고 시작합니다. 다음으로 다음 명령을 실행합니다.

!10G

!는 Vim에 필터 작업을 수행 중임을 알려줍니다. Vim 편집기는 파일의 어느 부분을 필터링할지를 나타내는 모션 명령이 뒤따를 것으로 예상합니다. 10G 명령은 Vim에 10번째 줄로 이동하라고 알려주므로, 이제 1번째 줄(현재 줄)에서 10번째 줄(10G)까지 필터링해야 한다는 것을 알게 됩니다.

필터링을 예상하면서 커서가 화면 하단으로 떨어지고 ! 프롬프트가 표시됩니다. 이제 필터 프로그램의 이름(이 경우 sort)을 입력할 수 있습니다.

따라서 전체 명령은 다음과 같습니다.

**!10Gsort<Enter>**

결과적으로 정렬 프로그램은 처음 10줄에서 실행됩니다. 프로그램의 출력은 이러한 줄을 대체합니다.

**!!** 명령은 현재 줄을 필터로 실행합니다. (저는 이것이 시스템 명령의 출력을 파일에 넣는 좋은 방법이라는 것을 알았습니다.) 예를 들어, readme.txt 파일을 편집하고 현재 디렉토리의 파일 목록을 포함하려고 합니다. 커서를 빈 줄에 놓고 다음을 입력합니다.

**!!ls**

이렇게 하면 ls 명령의 출력이 내 파일에 저장됩니다.

또 다른 요령은 변경 사항에 타임스탬프를 찍는 것입니다. 현재 날짜 시간을 얻으려면(UNIX에서) 다음 명령을 사용합니다.

**!!date**

이것은 변경 내역 등에 매우 유용합니다.

참고: 이렇게 !!를 사용하는 것은 기술적으로 필터링이 아닙니다. ls 및 date와 같은 명령은 표준 입력을 읽지 않기 때문입니다.

## Editing Another File

# A. Installing Vim

www.vim.org 웹사이트에서 VIM을 다운로드할 수 있습니다. 이 사이트에는 Vim 소스는 물론 다양한 시스템에 맞춰 미리 컴파일된 바이너리도 들어 있습니다.

## UNIX

www.vim.org에서 다양한 UNIX 시스템에 대한 사전 컴파일된 바이너리를 얻을 수 있습니다. www.vim.org로 이동하여 "Vim 다운로드" 링크를 클릭한 다음 "바이너리 페이지" 링크를 따르세요.

$ sudo apt install vim-gtk3