Two-Config Keybinding Guide (Expanded & Explained)

Part A: IdeaVim in JetBrains (Leader: **,**) · Part B: Vim/Neovim (Leader: **<Space>**)

Part A — IdeaVim (JetBrains) — Leader: ,

This section documents every mapping and option from the JetBrains/IdeaVim configuration, with rationale and examples.

Core Options & Plugins

Setting	Effect / Rationale
clipboard+=unnamedplus	Use the system clipboard for all yanks/deletes/puts; smoother copy/paste with OS.
ideajoin	Improves `J` (join) to follow IDE formatter rules to avoid broken code style.
highlightedyank	Briefly highlight yanked text for feedback.
surround	Enable `surround` motions: add/change/delete surrounding quotes/brackets quickly.
easymotion	Jump across the screen with a two■char hop (`s` below).
matchit	Smarter `%` matching for (), {}, [], HTML/XML tags.
notimeout	Disable key■chord timeout; helpful for long leader sequences.

Better Behaviors

Key	Behavior
Key	Behavior / Reason
<C-o>	Back in IDE navigation history (like Browser Back).
<C-i>	Forward in IDE navigation history.
K	Show hover info / quick documentation at caret.

Blocked Shortcuts (avoid accidents)

Chord	Why disable?
<A-S-f>, <C-S-f>	Disable large global actions (e.g., “Format File” or “Find in path” accidents).
<C-n>/<C-p> in Insert and Normal	Prevent conflicts with completion or external tools; rely on mapped actions instead.

Plugin Shortcuts

Key	Action
nmap s / xmap s	EasyMotion hop in normal/visual for two■character jumps.
nmap gm / xmap gm	Matchit: jump to matching bracket/tag; works in visual too.

CamelCase Text■Objects

Key	What it operates on
vic / cic / dic / yic	Operate on “inside camelCase component”: select/change/delete/yank.
nnoremap ci → "_ci	Force `ci` to use black■hole register so CamelCase plugin doesn’t override registers.

Argument & Chain Splitters (Leader ,sf* / ,sc* / ,st)

Key	Description + Example
,sf2	Split first 2 function arguments to one per line; remaining stay inline. Great before formatting.

```
Before: foo(a, b, c, d, e, f, g)
Press:  ,sf2
After:
    foo(
        a,
        b,
        c, d, e, f, g
    )
```

,sf3	Split first 3 function arguments to one per line; remaining stay inline. Great before formatting.

```
Before: foo(a, b, c, d, e, f, g)
Press:  ,sf3
After:
    foo(
        a,
        b,
        c,
        d, e, f, g
    )
```

,sf4	Split first 4 function arguments to one per line; remaining stay inline. Great before formatting.

```
Before: foo(a, b, c, d, e, f, g)
Press:  ,sf4
After:
    foo(
        a,
        b,
        c,
        d,
        e, f, g
    )
```

,sf5

Split first 5 function arguments to one per line; remaining stay inline. Great before formatting.

```
Before: foo(a, b, c, d, e, f, g)
Press:  ,sf5
After:
    foo(
        a,
        b,
        c,
        d,
        e,
        f, g
    )
```

,sf6

Split first 6 function arguments to one per line; remaining stay inline. Great before formatting.

```
Before: foo(a, b, c, d, e, f, g)
Press:  ,sf6
After:
    foo(
        a,
        b,
        c,
        d,
        e,
        f,
        g
    )
```

,sf7

Split first 7 function arguments to one per line; remaining stay inline. Great before formatting.

```
Before: foo(a, b, c, d, e, f, g)
Press:  ,sf7
After:
    foo(
        a,
        b,
        c,
        d,
        e,
        f,
        g,
    )
```

Key	Description + Example

,sc2

Split chained calls so the first 2 are stacked; rest remain chained on last line.

```
Before: obj.first().second().third().fourth().fifth().sixth()
Press:  ,sc2
After:
    obj
        .first()
        .second()
        .third().fourth().fifth().sixth()
```

,sc3

Split chained calls so the first 3 are stacked; rest remain chained on last line.

```
Before: obj.first().second().third().fourth().fifth().sixth()
Press:  ,sc3
After:
    obj
        .first()
        .second()
        .third()
        .fourth().fifth().sixth()
```

,sc4

Split chained calls so the first 4 are stacked; rest remain chained on last line.

```
Before: obj.first().second().third().fourth().fifth().sixth()
Press:  ,sc4
After:
    obj
        .first()
        .second()
        .third()
        .fourth()
        .fifth().sixth()
```

,sc5

Split chained calls so the first 5 are stacked; rest remain chained on last line.

```
Before: obj.first().second().third().fourth().fifth().sixth()
Press:  ,sc5
After:
    obj
        .first()
        .second()
        .third()
        .fourth()
        .fifth()
        .sixth()
```

,sc6

Split chained calls so the first 6 are stacked; rest remain chained on last line.

```
Before: obj.first().second().third().fourth().fifth().sixth()
Press:  ,sc6
After:
    obj
        .first()
        .second()
        .third()
        .fourth()
        .fifth()
        .sixth()
```

,sc7

Split chained calls so the first 7 are stacked; rest remain chained on last line.

```
Before: obj.first().second().third().fourth().fifth().sixth()
Press:  ,sc7
After:
    obj
        .first()
        .second()
        .third()
        .fourth()
        .fifth()
        .sixth()
```

Key	Description + Example

,st	Split a `? :` ternary across lines for clarity and future diffs.

```
Before: result = condition ? valueA : valueB
Press:  ,st
After:
    result = condition
        ? valueA
        : valueB
```

Config Maintenance

Key	Action
,ei	Open `~/.ideavimrc` in editor.
,si	Reload `~/.ideavimrc` (source).

IDE Navigation & Refactoring

Key	Action / Context
gs / gc	Go to Symbol / Class (IDE search palette).
<C-p> / <C-e>	Go to File / Recent Files.
,fu	Find Usages for symbol under caret.
,gr / ,gi	Go to Related / Go to Implementation.
,ff	Find in Path (project■wide search).
,re	Rename element (safe refactor).
,ao	Select all occurrences (multi■cursor).
,rp	Replace.
,il / ,iv	Inline / Introduce variable.
<S-Space>	Call inline completion (IDE feature).
,oe	Reveal in (project/file explorer).
,os	Recent projects list.
,cs	Close current project.
,ta	Find Action by ID (handy for discovering action names).
,og	Open Terminal: `wt lg` (Windows Terminal alias).
,ie / ,oa	Sweep AI: show prompt bar / new chat.

Save / Cleanup / Imports

Key	Action
<C-s> (also in Insert)	Save all files.

,fd	Reformat code (IDE formatter).
,cc	Silent code cleanup (IDE profile).
,oi	Optimize imports.

Tool Windows & Popups

Key	Action
,sp / ,op / ,of / ,ot / ,od / ,oc	Select in Project / Project TW / Find TW / Terminal TW / Debug TW / Commit
,hw	Hide all tool windows.
<C-m>	Open Popup menu.
,pi / ,fs / ,rf	Parameter Info / File Structure / Refactorings Quick List.

Editor Navigation & Tabs

Key	Action
,ne / ,pe	Next/Previous error at caret scope.
[d /]d	ReSharper: prev/next error in solution.
[f /]f	Method up/down.
<C-h> / <C-l>	Previous/Next tab (also in Insert).
,tc / ,to / ,tp / ,tm	Close / Close others / Pin / Close unmodified tabs.

Window (Split) Management — Vim Motions

Key	Action
,wc / ,wo	Close current split / Only keep this split.
,wj / ,wk / ,wh / ,wl	Move focus (down/up/left/right).
,ws / ,wv	Horizontal / Vertical split.

Build / Run / Debug

Key	Action
,ba / ,ra / ,da / ,sa	Build solution / Run / Debug / Stop.
,rt / ,dt	Run / Debug tests in context.
,tb	Toggle line breakpoint.
<C-S-A-j/k/h/l>	Step Over / Resume / Step Out / Step Into.
,ee	Quick Evaluate Expression (debugger).

Keyboard Handlers (Route chords to IdeaVim)

A long list of `sethandler` rules ensures Control/Alt chords go to Vim first. This prevents the IDE from intercepting them, preserving consistent modal behavior.

AI Completion

Key	Action
<Tab>	Accept Sweep AI inline completion when shown.

Part B — Vim / Neovim — Leader:

This section explains the native Vim mappings. Many are quality■of■life tweaks focused on repeatability and safety.

Leader & Core Options

Setting	Effect / Rationale
Leader = <Space>	Disable bare <Space> (`<Nop>`) then use it as a Leader prefix for readable combos.
clipboard+=unnamed	Also sync unnamed register with OS clipboard (X11/mac pbcopy dependent).
scrolloff=10	Keep 10 context lines above/below cursor for stable visual focus.
incsearch + hlsearch	Incremental search preview; highlight all matches after search.
number + relativenumber	Hybrid line numbers: absolute for current line, relative for motions.
ignorecase + smartcase	Case■insensitive search unless pattern has uppercase.

Better Navigation & Editing Behaviors

Key	Behavior / Why
<C-d>zz / <C-u>zz	Half■page down/up then center cursor (`zz`) so target line stays in view.
n / N → nzz / Nzz	Next/previous search result then center; keeps context.
Y → y$	Yank to end of line (more intuitive than Vim’s default `yy`).
J → Jh	Join lines but move one char left first to avoid eating leading space.
x / s / c / C use "_ (black hole)	Delete/change without clobbering default register.
Visual p uses "_dP	Replace selection with paste without yanking the replaced text.
Visual < / > keep selection	`<gv`/`>gv` reselects after indent so you can indent repeatedly.

Productivity Shortcuts

Key	Action / Rationale
<C-j> / <C-k> (normal)	Map to `<C-e>`/`<C-y>`: smooth scroll down/up without moving cursor.
<C-j> / <C-k> (insert)	Next/previous completion item (<C-n>/<C-p>) for fast completion.
gh / gl (all modes where mapped)	`^` / `$`: start or end of line, easy on non■US keyboards.
+ / -	Increment/decrement number under cursor (arithmetic on integers).
U	Redo (`<C-r>`), for a symmetric undo/redo mnemonic.
Q	Replay macro in register `q` (`@q`) quickly.
[[/]]	Jump to previous/next section (useful in code blocks with braces).

Leader Utilities & Misc

Key	Action
<Space>ns	Clear search highlights (`:nohlsearch`).

Prevent Bad Habits

Key	Why
Arrow keys disabled	Encourages hjkl muscle memory and Vim■native motions.
<C-n>/<C-p> in Insert disabled (also Normal <C-n>)	Avoids register clobbering or unwanted completion when not intentional.

Edit & Reload Your Vim Config

Key	Action
<Space>ev	Open `~/.vimrc`.
<Space>sv	Reload `~/.vimrc` (source).

Notes & Tips

• All delete/change mappings that use the black■hole register ("_) preserve your last yank, which is ideal for repetitive edits. • The split helpers in Part A are intentionally simple keystroke macros; running the IDE formatter afterward will fix indentation. • Consider pairing the arrow■key lockout with `which-key` or a cheatsheet to ease the transition if you’re new to leader maps.