

```

/*
8kun Formatting Tools
Version 2.1.0
    - Adds a toolbar above every post textarea (including quick reply)
with 8kun's formatting options
    - Press Esc to close the quick-reply window
    - Press c (outside of text fields) to go to the catalog
    - Hover over each button to see its keyboard shortcut
    - Formatting keeps native undo (Ctrl+Z) where the browser supports it
You can add your own toolbar items by modifying 'const formats'. Syntax:
    format_name: {
        displayText: 'string',          text shown on the toolbar button
        altText: 'string',              optional, tooltip text
        options: {
            prefix: 'string',            prepended to the selection
            suffix: 'string',            appended to the selection
            multiline: true/false,       optional, may span line breaks
            (default false)
            exclusiveLine: true/false    optional, formatted text must be on
its own line (default false)
        },
        edit: function (box, options)    runs when the button is pressed
        shortcutKey: 'string'            optional, Ctrl/Cmd + key shortcut
    }
*/
(function () {
    'use strict';

    const CSS = `
/* generated by 8kun Formatting Tools */
.tf-toolbar { padding: 0 5px 1px 5px; }
.tf-toolbar a { text-decoration: none; cursor: pointer; }
.tf-toolbar a[data-format="bold"] { font-weight: bold; }
.tf-toolbar a[data-format="italics"] { font-style: italic; }
.tf-toolbar a[data-format="under"] { text-decoration: underline; }
.tf-toolbar a[data-format="code"] { font-family: "Courier New", Courier,
monospace; }
.tf-toolbar a[data-format="strike"] { text-decoration: line-through; }
.tf-toolbar a[data-format="heading"] { color: #AF0A0F; font-weight:
bold; }
`;

    const wrap = function (box, options) { wrapSelection(box, options); };

    const formats = {
        bold: { displayText: 'B',          altText: 'bold',
options: { prefix: "''", suffix: "''" }, edit: wrap, shortcutKey:
'b' },
        italics: { displayText: 'i',          altText: 'italics',
options: { prefix: "'", suffix: "'" }, edit: wrap, shortcutKey:
'i' },
        under: { displayText: 'U',          altText: 'underline',
options: { prefix: '__', suffix: '__' }, edit: wrap, shortcutKey:
'u' },

```

```

    spoiler: { displayText: 'spoiler', altText: 'mark as spoiler',
options: { prefix: '[spoiler]', suffix: '[/spoiler]' }, edit: wrap,
shortcutKey: 's' },
    code: { displayText: 'code', altText: 'code formatting',
options: { prefix: '[code]', suffix: '[/code]', multiline: true }, edit:
wrap },
    strike: { displayText: 'strike', altText: 'strikethrough',
options: { prefix: '~~', suffix: '~~' }, edit: wrap },
    heading: { displayText: 'heading', altText: 'redtext',
options: { prefix: '==', suffix: '==', exclusiveLine: true }, edit:
wrap }
};

```

```

const qs = (sel, parent = document) => parent.querySelector(sel);
const qsa = (sel, parent = document) =>
Array.from(parent.querySelectorAll(sel));
const isLineBreak = ch => ch === '\r' || ch === '\n';

```

```

function initCSS() {
    if (document.getElementById('tf-toolbar-css')) return;
    const style = document.createElement('style');
    style.id = 'tf-toolbar-css';
    style.textContent = CSS;
    document.head.append(style);
}

```

// Replace a range of the textarea, preserving undo history when possible.

```

function replaceRange(box, start, end, text) {
    box.focus();
    box.setSelectionRange(start, end);
    let ok = false;
    try { ok = document.execCommand('insertText', false, text); } catch
(e) { ok = false; }
    if (!ok) {
        box.setRangeText(text, start, end, 'end');
        // let 8kun's own scripts (char counter, etc.) see the change
        box.dispatchEvent(new Event('input', { bubbles: true }));
    }
}

```

```

function wrapSelection(box, options) {
    if (!box) return;
    const { prefix = '', suffix = '', multiline = false, exclusiveLine =
false } = options;
    const scrollTop = box.scrollTop;
    const text = box.value;
    let start = box.selectionStart;
    let end = box.selectionEnd;

    // Expand to whole lines first, then trim, so no text gets reordered.
    if (exclusiveLine) {
        while (start > 0 && !isLineBreak(text[start - 1])) start--;
        while (end < text.length && !isLineBreak(text[end])) end++;
    }
}

```

```

    }
    // Leave leading/trailing spaces outside the markup (e.g. from
double-click selection)
    while (end > start && text[end - 1] === ' ') end--;
    while (start < end && text[start] === ' ') start++;

    const selected = text.slice(start, end);
    let replacement;
    if (multiline || !/[\r\n]/.test(selected)) {
        replacement = prefix + selected + suffix;
    } else {
        // Wrap each non-empty line separately; keep blank lines untouched.
        replacement = selected.split(/(\r\n|\r|\n)/).map((part, i) =>
            (i % 2 === 1 || part.trim() === '') ? part : prefix + part +
suffix
        ).join('');
    }

    replaceRange(box, start, end, replacement);

    // Empty selection: caret goes inside the markup. Otherwise: after
it.
    const caret = selected.length === 0 ? start + prefix.length : start +
replacement.length;
    box.setSelectionRange(caret, caret);
    box.scrollTop = scrollTop;
}

function buildToolbar() {
    const toolbar = document.createElement('div');
    toolbar.className = 'tf-toolbar';
    for (const [name, fmt] of Object.entries(formats)) {
        if (fmt.displayText === null) continue;
        if (toolbar.childNodes.length) toolbar.append(' | ');
        const a = document.createElement('a');
        a.href = '#';
        a.setAttribute('role', 'button');
        a.dataset.format = name;
        a.textContent = fmt.displayText;
        if (fmt.altText) {
            a.title = fmt.altText + (fmt.shortcutKey ? `
(Ctrl+${fmt.shortcutKey.toUpperCase()})` : '');
        }
        toolbar.append(a);
    }
    return toolbar;
}

// Attach a toolbar to every post textarea that doesn't have one yet
// (covers the main form and quick reply, which is created later).
function attachToolbars() {
    for (const textarea of qsa('textarea[name="body"]')) {
        const prev = textarea.previousElementSibling;
        if (prev && prev.classList.contains('tf-toolbar')) continue;

```

```

        textarea.parentElement.insertBefore(buildToolbar(), textarea);
    }
}

function textareaForToolbar(toolbar) {
    const next = toolbar.nextElementSibling;
    if (next && next.matches('textarea')) return next;
    return qs('textarea[name="body"]', toolbar.parentElement);
}

function isTypingTarget(el) {
    return el && (el.isContentEditable ||
/^ (INPUT|TEXTAREA|SELECT)$/.test(el.tagName));
}

function main() {
    if (window.active_page !== 'thread' && window.active_page !==
'index') return;
    if (!qs('textarea[name="body"]')) return;
    initCSS();
    attachToolbars();

    // Watch for quick reply (and any other late-added forms)
    let pending = false;
    new MutationObserver(() => {
        if (pending) return;
        pending = true;
        requestAnimationFrame(() => { pending = false;
attachToolbars(); });
    }).observe(document.body, { childList: true, subtree: true });

    // Toolbar buttons
    document.addEventListener('click', e => {
        const btn = e.target.closest && e.target.closest('.tf-toolbar
a[data-format]');
        if (!btn) return;
        e.preventDefault();
        const fmt = formats[btn.dataset.format];
        const box = textareaForToolbar(btn.parentElement);
        if (fmt && box) {
            fmt.edit(box, fmt.options);
            box.focus();
        }
    });

    // Keyboard handling
    document.addEventListener('keydown', e => {
        const t = e.target;

        if (t.matches && t.matches('textarea[name="body"]')) {
            // Esc closes quick reply
            if (e.key === 'Escape' && t.closest('#quick-reply')) {
                const closeBtn = qs('#quick-reply .close-btn');
                if (closeBtn) closeBtn.click();
            }
        }
    });
}

```

```

        return;
    }
    // Ctrl/Cmd + key shortcuts
    if ((e.ctrlKey || e.metaKey) && !e.altKey && !e.shiftKey &&
e.key) {
        const key = e.key.toLowerCase();
        const fmt = Object.values(formats).find(f => f.shortcutKey ===
key);
        if (fmt) {
            e.preventDefault();
            fmt.edit(t, fmt.options);
        }
    }
    return;
}

    // "c" jumps to the catalog, but never while typing somewhere
    if (e.key === 'c' && !e.ctrlKey && !e.altKey && !e.metaKey
&& !e.shiftKey
        && !isTypingTarget(t) && window.board_name) {
        location.href =
`//${location.host}/${window.board_name}/catalog.html`;
    }
    });
}

if (document.readyState === 'loading') {
    document.addEventListener('DOMContentLoaded', main);
} else {
    main();
}
})();

/*
 * q.js v2018.3-6.2 (patched)
 * based on http://anonsw.github.io/8chjs/
 */

// User Settings
var anonsw = {
    qflair: '', // Examples: REAL, &rarr;
    qtrip: '!!Hs1Jq13jV6', // tripcode used to identify Q posts
    qcolor: '#99d6ff',
    youcolor: '#ffff00',
    scrollcolor: 'rgba(153, 153, 153, 0.6)',
    scrollbackcolor: '#333',
    scrolltime: 400, // ms
    updateDelay: 200, // ms
    sidenavWidth: 30, // px

    nopeEnabled: true, // "Nope" button to permanently hide an image
    by its hash

```

```

    floodEnabled: false,    // default; the toggle link remembers your
last choice
    floodThreshold: 15,     // min # posts before beginning fade
    floodVanish: 25,       // max # posts before completed fade/hide
    floodBehavior: 'fade', // hide, fade
    fadenametripregex: /^(Anon(ymous)?-.*|.*-!!Hs1Jq13jV6)$/i,
    fadenametripfloodvalue: -1, // Effective post count for fading, or -1
for auto of floodThreshold+post count
    strikeThroughNameFags: true,

    rateHistoryLen: 50,     // Data points on chart
    rateAvgLen: 10         // Number of data points to average for
instantaneous rate
};

(function (anonsw_main, $) {
    'use strict';
    if (!$) return;

    // ----- helpers -----
    function debounce(fn, ms) {
        var t;
        return function () {
            var self = this, args = arguments;
            clearTimeout(t);
            t = setTimeout(function () { fn.apply(self, args); }, ms);
        };
    }

    function rafThrottle(fn) {
        var queued = false;
        return function () {
            if (queued) return;
            queued = true;
            requestAnimationFrame(function () { queued = false; fn(); });
        };
    }

    function boardlist() { return $('div.boardlist').first(); }

    // Case-insensitive :icontains selector (works on jQuery 1.x-3.x)
    var pseudos = $.expr.pseudos || $.expr[':'];
    if (!pseudos.icontains) {
        pseudos.icontains = $.expr.createPseudo(function (arg) {
            var needle = String(arg).toUpperCase();
            return function (el) { return
$(el).text().toUpperCase().indexOf(needle) >= 0; };
        });
    }

    function immediateText(el) {
        return el.contents().not(el.children()).text();
    }

```

```

function myScrollTo(el) {
    if (!el) return;
    $('html, body').stop().animate({
        scrollTop: $(el).offset().top - boardlist().outerHeight()
    }, anonsw.scrolltime);
}

// ----- image blacklist ("Nope") -----
var imageBlacklist = new Set();

function loadImageBlacklist() {
    try {
        JSON.parse(localStorage.getItem('imageBlacklist') ||
'[]').forEach(function (md5) {
            if (md5) imageBlacklist.add(md5);
        });
    } catch (e) { /* corrupt or unavailable storage */ }
}

function saveImageBlacklist() {
    try { localStorage.setItem('imageBlacklist',
JSON.stringify(Array.from(imageBlacklist))); } catch (e) {}
}

function removeBlacklistedImages() {
    if (!imageBlacklist.size) return;
    $('img.post-image').each(function () {
        var md5 = this.getAttribute('data-md5');
        if (md5 && imageBlacklist.has(md5)) this.remove();
    });
}

function addNopeButtons() {
    $('div.post').not('.post-hover').each(function () {
        var $post = $(this);
        if ($post.children('.nope').length) return;
        if (!$post.find('img.post-image').length
&& !$post.hasClass('op')) return;
        $post.prepend('<input type="button" class="nope" value="Nope"
title="Hide these images everywhere">');
    });
}

$(document).on('click', 'input.nope', function (e) {
    e.preventDefault();
    e.stopPropagation();
    loadImageBlacklist(); // merge with changes made in other tabs
    var $post = $(this).closest('.post');
    // OP images live outside div.post, in the thread's div.files
    var $imgs = $post.find('img.post-image');
    if ($post.hasClass('op')) $imgs =
$imgs.add($post.closest('div.thread').children('div.files').find('img.pos
t-image'));
    $imgs.each(function () {

```

```

        var md5 = this.getAttribute('data-md5');
        if (md5) imageBlacklist.add(md5);
        this.remove();
    });
    saveImageBlacklist();
    removeBlacklistedImages();
});

// Keep tabs in sync
window.addEventListener('storage', function (e) {
    if (e.key === 'imageBlacklist') { loadImageBlacklist();
removeBlacklistedImages(); }
});

// ----- navigation state -----
var nav = {
    q: { posts: [], idx: -1 },
    you: { posts: [], idx: -1 }
};
var ctx = null, borderSz = 1, scrollWd = 0;

function go(kind, where) {
    var n = nav[kind];
    if (!n || !n.posts.length) return;
    if (where === 'first') n.idx = 0;
    else if (where === 'last') n.idx = n.posts.length - 1;
    else if (where === 'next') n.idx = Math.min(n.idx + 1,
n.posts.length - 1);
    else if (where === 'prev') n.idx = Math.max(n.idx - 1, 0);
    myScrollTo(n.posts[n.idx]);
}

// Backwards-compatible public API (anonsw_main.nextq(), etc.)
['first', 'prev', 'next', 'last'].forEach(function (w) {
    anonsw_main[w + 'q'] = function () { go('q', w); };
    anonsw_main[w + 'you'] = function () { go('you', w); };
});

function navHtml(label, countCls, kind) {
    return '<span class="anonsw-nav">[ ' +
        '<a href="#" data-anonsw="first" + kind + ' " title="First"><i
class="fa fa-step-backward"></i></a> ' +
        '<a href="#" data-anonsw="prev" + kind + ' "
title="Previous"><i class="fa fa-backward"></i></a> ' +
        '<span style="filter:brightness(70%);">' + label + '</span> '
+
        '<span class="' + countCls + '">(?:?)</span> ' +
        '<a href="#" data-anonsw="next" + kind + ' " title="Next"><i
class="fa fa-forward"></i></a> ' +
        '<a href="#" data-anonsw="last" + kind + ' " title="Last"><i
class="fa fa-step-forward"></i></a>' +
        ' ]</span>';
}

```

```

function insertNav() {
    var $bl = $('div.boardlist');
    $bl.append(navHtml('Q', 'qcount', 'q'));
    $bl.append(navHtml(' (You)', 'youcount', 'you'));
    $bl.append('<span class="anonsw-nav">[ <a href="#" data-
anonsw="toggleFlood">Turn Post Fading ' +
        '<span class="toggleFloodState">' + (anonsw.floodEnabled ?
'Off' : 'On') + '</span></a> ]</span>');
    $bl.append('<span class="anonsw-nav">[ Posts: <span
class="anonsw-postcount">?</span> | ' +
        'Post Rate: <span class="postRate">0.0</span> posts/min ' +
        '<canvas class="postRateChart"></canvas>]</span>');

    $('<div>
    .postRate, .anonsw-postcount').css('color', $('div.boardlist
a').first().css('color'));
    var height = Math.max(boardlist().height() - 1, 10);
    $('<div>
    .postRateChart').each(function () {
        $(this).css({ width: '100px', height: height + 'px',
'vertical-align': 'middle' });
        this.height = 20;
        this.width = 100;
    });
}

$(document).on('click', '[data-anonsw]', function (e) {
    e.preventDefault();
    var action = $(this).attr('data-anonsw');
    if (action === 'toggleFlood') { toggleFlood(); return; }
    var m = /^(first|prev|next|last)(q|you)$/.exec(action);
    if (m) go(m[2], m[1]);
});

function insertSidenav() {
    var $nav = $('<div>
    <canvas id="sidenav" title="Click to
jump"></div>').appendTo('body');
    $nav.css({
        position: 'fixed',
        top: boardlist().outerHeight() || 0,
        right: 0,
        width: anonsw.sidenavWidth,
        height: $(window).height() - (boardlist().outerHeight() ||
0),
        'background-color': anonsw.scrollbackcolor,
        cursor: 'pointer',
        'z-index': 50
    });
    $('<div>
    body').css('margin-right', anonsw.sidenavWidth);
    ctx = $nav[0].getContext('2d');
    ctx.canvas.height = 2048;
    ctx.canvas.width = anonsw.sidenavWidth;
    scrollWd = ctx.canvas.width / 2;

    // Click anywhere on the bird's-eye view to jump there
    $nav.on('click', function (e) {

```

```

        var rect = this.getBoundingClientRect();
        var frac = (e.clientY - rect.top) / rect.height;
        $('html, body').stop().animate({
            scrollTop: frac * $(document).height() -
$(window).height() / 2
        }, anonsw.scrolltime);
    });
}

// ----- nav updates -----
// "Current" = the last post whose top edge has scrolled past the
middle
// of the screen. Before reaching the first one, the position is 0.
// At the very bottom of the page, any post that is on screen counts.
function currentIndex(posts) {
    if (!posts.length) return -1;
    var scrollTop = $(window).scrollTop();
    var winH = $(window).height();
    var atBottom = scrollTop + winH >= $(document).height() - 2;
    var line = atBottom ? scrollTop + winH : scrollTop + winH / 2;
    var idx = -1;
    for (var i = 0; i < posts.length; i++) {
        if ($(posts[i]).offset().top <= line) idx = i;
        else break; // posts are in document order
    }
    return idx;
}

// Total posts in the thread. Uses 8kun's stats counter when present,
// otherwise counts the posts on the page (OP + replies, no hover
previews).
function threadPostCount() {
    var n = parseInt($('#thread_stats_posts').first().text(), 10);
    if (!isNaN(n)) return n;
    if (window.active_page !== 'thread') return NaN;
    return $('div.thread').first().find('div.post').not('.post-
hover').length;
}

function updateNavCounts() {
    nav.q.idx = currentIndex(nav.q.posts);
    nav.you.idx = currentIndex(nav.you.posts);
    $('.qcount').text('(' + (nav.q.idx + 1) + ':' +
nav.q.posts.length + ')');
    $('.youcount').text('(' + (nav.you.idx + 1) + ':' +
nav.you.posts.length + ')');
    var total = threadPostCount();
    $('.anonsw-postcount').text(isNaN(total) ? '?' : total);
}

function drawMarkers(posts, color, docH) {
    var minheight = ctx.canvas.height / Math.max(1,
$(window).height() - boardlist().outerHeight());
    ctx.fillStyle = color;

```

```

        for (var i = 0; i < posts.length; i++) {
            var $el = $(posts[i]);
            var h = Math.max($el.outerHeight() / docH *
ctx.canvas.height, minheight);
            var y = $el.offset().top / docH * ctx.canvas.height;
            ctx.fillRect(borderSz, y, ctx.canvas.width - borderSz * 2,
h);
        }
    }

function updateNavGraphics() {
    if (!ctx) return;
    var blH = boardlist().outerHeight() || 0;
    var winH = $(window).height();
    var docH = Math.max(1, $(document).height());
    $('#sidenav').css({ top: blH, height: winH - blH });
    ctx.clearRect(0, 0, ctx.canvas.width, ctx.canvas.height);

    drawMarkers(nav.q.posts, anonsw.qcolor, docH);
    drawMarkers(nav.you.posts, anonsw.youcolor, docH);

    // Viewport window
    ctx.fillStyle = anonsw.scrollcolor;
    ctx.fillRect(
        scrollWd / 2,
        $(window).scrollTop() / docH * ctx.canvas.height,
        scrollWd,
        winH / docH * ctx.canvas.height
    );

    // Red marker at the bottom when thread is past 750 posts
    if (threadPostCount() > 750) {
        var barHeight = (4 * ctx.canvas.height) / Math.max(1, winH -
blH);
        ctx.fillStyle = '#f66';
        ctx.fillRect(0, ctx.canvas.height - barHeight,
ctx.canvas.width, barHeight);
    }
}

function updateNavPostRate() {
    var posts = $('div.post').not('.post-hover');
    var avgLen = anonsw.rateAvgLen;
    var startPost = Math.max(1, posts.length - (anonsw.rateHistoryLen
+ avgLen) + 1);
    var ratehistory = [];
    var timehistory = [];

    for (var i = startPost; i < posts.length; i++) {
        var t = parseInt($(posts[i]).find('.intro
time').first().attr('unixtime'), 10);
        if (isNaN(t)) continue;
        timehistory.push(t);
        var n = timehistory.length;

```

```

        if (n - avgLen - 1 >= 0) {
            var minutes = (timehistory[n - 1] - timehistory[n -
avgLen - 1]) / 60;
            ratehistory.push(avgLen / Math.max(minutes, 1 / 60)); //
avoid divide-by-zero
        } else {
            ratehistory.push(0);
        }
    }

    $('.postRate').text(ratehistory.length ?
ratehistory[ratehistory.length - 1].toFixed(1) : '0.0');

    $('.postRateChart').each(function () {
        var gctx = this.getContext('2d');
        gctx.clearRect(0, 0, gctx.canvas.width, gctx.canvas.height);
        if (ratehistory.length <= avgLen + 1) return;

        var maxRate = Math.max.apply(null, ratehistory);
        var minRate = Math.min.apply(null, ratehistory);
        if (minRate > maxRate - 0.5) { minRate = maxRate - 0.5;
maxRate = maxRate + 0.5; }
        if (minRate < 0) minRate = 0;
        var maxTime = timehistory[timehistory.length - 1];
        var minTime = timehistory[avgLen];
        var span = Math.max(1, maxTime - minTime);
        var yOf = function (r) { return gctx.canvas.height - (r -
minRate) / (maxRate - minRate) * gctx.canvas.height; };

        gctx.strokeStyle = $('div.boardlist a').first().css('color');
        gctx.beginPath();
        gctx.moveTo(0, yOf(ratehistory[avgLen]));
        for (var j = avgLen + 1; j < ratehistory.length; j++) {
            gctx.lineTo((timehistory[j] - minTime) / span *
gctx.canvas.width, yOf(ratehistory[j]));
        }
        gctx.stroke();
    });
}

function updateNav() {
    updateNavCounts();
    updateNavGraphics();
    updateNavPostRate();
}

// ----- post classification -----
function setQPosts() {
    nav.q.posts = $('div.post:not(.post-hover) > p.intro > label >
span.trip')
        .filter(function () { return
$(this).text().indexOf(anonsw.qtrip) !== -1; })
        .closest('div.post')
        .get();
}

```

```

    }

    function highlightQ() {
        $(nav.q.posts).each(function () {
            var $p = $(this);
            if ($p.hasClass('anonsw-q')) return; // only decorate once
            $p.addClass('anonsw-q').css('background-color',
anonsw.qcolor);
            if (anonsw.qflair) {
                $p.find('> p.intro > label >
span.trip').first().prepend(anonsw.qflair + ' ');
            }
        });
    }

    function removeInvalidYous() {
        $('div.body:icontains("(You)")').each(function () {
            $(this).find(':not(small)').contents().filter(function ()
{ return this.nodeType === 3; }).each(function () {
                if (/\\(\\+ *You *\\)+/i.test(this.textContent)) {
                    this.textContent = this.textContent.replace(/\\(\\+ *You
*\\)+/ig, 'you');
                }
            });
        });
    }

    function setYouPosts() {
        nav.you.posts = $(
            'div.post:not(.post-hover) > p.intro > label span.name >
span.own_post, ' +
            'div.post:not(.post-hover) > div.body > p.body-line >
small:icontains("(You)") '
        ).closest('div.post').get(); // closest() also de-duplicates
    }

    function highlightYous() {
        $(nav.you.posts).addClass('anonsw-you').css('background-color',
anonsw.youcolor);
    }

    // ----- flood fading -----
    function clearFlood() {
        $('div.post').off('.anonswflood').css('opacity', '');
        $('div.post.anonsw-hidden').removeClass('anonsw-hidden').show();
        $('div.anonsw-struck').removeClass('anonsw-struck').css('text-
decoration', '');
    }

    function setFloodPosts() {
        if (!anonsw.floodEnabled) return;
        var stats = {};
        var firstId = null;
    }

```

```

        $('div.post').not('.post-hover').not('.you').each(function () {
            var $post = $(this);
            var id = $post.find('p > span.poster_id').first().text();
            if (firstId === null) { firstId = id; return; } // skip OP
            if (!id || id === firstId) return;

            var s = stats[id] || (stats[id] = { count: 0, namefag: false,
floodcount: 0, posts: [] });
            s.count++;
            s.posts.push(this);
            if (!s.namefag) {
                var name = immediateText($post.find('p > label
span.name').first());
                var trip = $post.find('p > label >
span.trip').first().text();
                s.namefag = !anonsw.fadenametripregex.test(name + '-' +
trip);
            }
            if (s.namefag) {
                s.floodcount = anonsw.fadenametripfloodvalue < 0
                    ? anonsw.floodThreshold + s.count
                    : anonsw.fadenametripfloodvalue;
            } else {
                s.floodcount = s.count;
            }
        });

$.each(stats, function (id, v) {
    var $posts = $(v.posts);
    if (!(v.floodcount > anonsw.floodThreshold || v.namefag)) {
        $posts.off('.anonswflood').css('opacity', '');
        return;
    }
    if (anonsw.strikeThroughNameFags && v.namefag) {
        $posts.each(function () {
            $(this).find('p > label
span.name').first().addClass('anonsw-struck').css('text-decoration',
'line-through');
        });
    }
    if (anonsw.floodBehavior === 'fade') {
        var range = Math.max(1, anonsw.floodVanish -
anonsw.floodThreshold);
        var intensity = (anonsw.floodVanish -
Math.min(v.floodcount, anonsw.floodVanish)) / range;
        intensity = Math.min(1, Math.max(0.1, intensity));
        // .off() first so handlers don't pile up on every update
        $posts.off('.anonswflood')
            .css('opacity', intensity)
            .on('mouseenter.anonswflood', function ()
{ $(this).stop(true).animate({ opacity: 1 }, anonsw.updateDelay); })
            .on('mouseleave.anonswflood', function ()
{ $(this).stop(true).animate({ opacity: intensity },
anonsw.updateDelay); });
    }

```

```

        } else if (anonsw.floodBehavior === 'hide' && v.count >=
anonsw.floodVanish) {
            $posts.addClass('anonsw-hidden').hide();
        }
    });
}

function toggleFlood() {
    anonsw.floodEnabled = !anonsw.floodEnabled;
    try { localStorage.setItem('anonswFloodEnabled',
anonsw.floodEnabled ? '1' : '0'); } catch (e) {}
    $('toggleFloodState').text(anonsw.floodEnabled ? 'Off' : 'On');
    if (anonsw.floodEnabled) runq(); else clearFlood();
}
anonsw_main.toggleFlood = toggleFlood;

// ----- main update -----
function runq() {
    var steps = [setQPosts, highlightQ, removeInvalidYous,
setYouPosts, highlightYous, setFloodPosts];
    if (anonsw.nopeEnabled) steps.push(addNopeButtons,
removeBlacklistedImages);
    steps.push(updateNav);
    // Run each step independently so one failure doesn't stop the
rest
    steps.forEach(function (fn) {
        try { fn(); } catch (e) { console.error('[q.js] ' + fn.name +
' failed:', e); }
    });
}

anonsw_main.Q = function () {
    if (anonsw_main._initialized) return;
    if (window.active_page !== 'thread' && window.active_page !==
'index') return;
    anonsw_main._initialized = true;

    try {
        var saved = localStorage.getItem('anonswFloodEnabled');
        if (saved !== null) anonsw.floodEnabled = saved === '1';
    } catch (e) {}

    $('<style>.post > input.nope{font-size:9px;padding:0 3px;margin:0
4px 0 0;float:left;}</style>').appendTo('head');

    loadImageBlacklist();
    if ($('#div.boardlist').length) insertNav();
    insertSidenav();
    runq();

    var scheduleRun = debounce(runq, anonsw.updateDelay);
    var observer = new MutationObserver(function (mutations) {
        for (var i = 0; i < mutations.length; i++) {

```

```

        if (mutations[i].type === 'childList' &&
mutations[i].addedNodes.length) {
            scheduleRun();
            return;
        }
    });
    $('div.thread').each(function () { observer.observe(this,
{ childList: true }); });

    $(window).on('scroll.anonsw', rafThrottle(function () {
        updateNavCounts();
        updateNavGraphics();
    }));
    $(window).on('resize.anonsw', debounce(updateNav,
anonsw.updateDelay));
    };

    $(anonsw_main.Q);
}(window.anonsw_main = window.anonsw_main || {}, window.jQuery));

```