일을 하다 보면 좋은 글과 도구, 참고 페이지가 손에 붙듯 쌓인다. 문제는 쌓이는 속도를 사람이 따라가지 못한다는 점이다. 공유를 위해 문서로 옮겨 적고, 분류하고, 링크를 검증하는 작업이 축적되면 어느 순간부터 귀찮음이 시스템을 이긴다. 그래서 나는 몇 해 전부터 링크 수집과 배포를 자동화해 왔다. 핵심은 간단하다. 브라우저에서 북마클릿으로 한 번에 저장하고, 태그와 메타데이터를 덧붙여 RSS 피드로 흘려보낸 다음, 그 피드를 여러 곳에서 재활용한다. 이렇게 구성하면 개인의 링크모음은 물론 팀용 사이트 주소모음, 주제별 큐레이션, 주간 뉴스레터까지 자연스럽게 이어진다.

아래는 실전에서 써 먹는 구성과 노하우다. 서비스 선택에는 정답이 없다. 다만 데이터 구조를 먼저 잡고, 그 구조를 RSS와 북마클릿으로 관통시켜 두면 어떤 도구를 바꿔도 큰 혼란이 없다.

왜 자동화가 필요한가

링크는 의외로 시간이 많이 든다. 저장과 분류만으로 끝나지 않는다. 제목이 바뀌거나 404가 뜨고, 중복 저장이 생기고, 팀에서 누구는 크롬 북마크에, 누구는 노션에, 또 누구는 슬랙에 흘려보낸다. 이렇게 분산된 링크는 나중에 찾기 어렵다. 반면 수집 - 정리 - 배포를 분리하고 자동화하면 몇 가지 이점이 생긴다.

첫째, 입력 지점이 하나가 된다. 모든 링크가 북마클릿을 거치면 뒤처리가 단순해진다. 둘째, RSS는 가벼운 표준이라 호환성이 넓다. 블로그, 뉴스레터, 디스코드, 슬랙, 노션 위젯까지 두루 연결된다. 셋째, 태그와 메모를 일관되게 붙여 두면 찾기 성능이 확 올라간다. 평소엔 모아 보고, 필요할 때 필터링한다.

전체 흐름을 먼저 그린다

구성은 보통 이렇게 나뉜다. 수집은 사람의 손을 빌리되 클릭 한 번으로 끝나야 한다. 정리는 저장소가 맡는다. 스포츠무료중계 적절한 필드와 규칙이 있다면 태그, 스코어, 소스 분류 같은 전처리는 자동으로 처리된다. 배포는 RSS가 중심이 된다. 피드 하나를 만들어도 좋고, 태그별로 피드를 나눠도 좋다. 최종 소비자는 사이트, 메일, 슬랙, 모바일 위젯 등 상황에 맞춰 고른다.

사이트 주소모음 페이지나 주간 링크모음 뉴스레터 같은 산출물은 이 흐름의 부산물이 된다. 손으로 정리하던 시간을 아껴 본질적인 큐레이션에 집중할 수 있다.

RSS의 역할과 설계 포인트

RSS는 정보를 전달하는 그릇이다. 무엇을 담을지, 어떻게 자주 흘려보낼지를 스스로 정하면 된다. 피드에 꼭 넣는 필드는 제목, 원본 링크, 설명, 발행 시각이다. 여기에 태그를 category로 넣고, 원문 도메인, 저장자, 평점 같은 커스텀 키를 description 끝에 약속된 포맷으로 실어둔다. 나중에 필터링이나 통계에서 유용하다.

피드를 만들 도구는 다양하다. 북마크 서비스인 Raindrop.io와 Pinboard는 태그별 RSS를 기본 제공한다. Notion은 데이터베이스를 RSS로 내보내는 공식 기능이 없지만, Notion API로 스크립트를 만들어 5분 간격으로 정적 RSS를 생성한 뒤 GitHub Pages에 올릴 수 있다. 스프레드시트도 마찬가지다. Google Apps Script로 열을 읽어 RSS XML을 뿌려 주면 된다.

구독 측면에서는 대부분의 이메일 서비스와 CMS가 RSS를 받아들인다. 워드프레스, 고스트, 미디엄, 텔레그램 봇, 슬랙, 디스코드, 노션 임베드까지 거부감이 없다. 마케팅 도구는 주로 UTM 파라미터를 원하지만, 큐레이션 용도라면 오히려 UTM을 제거해 원본 주소를 보존하는 편이 낫다. 취향의 문제라기보다 유지보수의 문제다. 리디렉션이 여러 겹 끼면 링크 검사에 자꾸 걸린다.

북마클릿이 핵심 동작을 만든다

북마클릿은 즐겨찾기 형태의 간단한 자바스크립트다. 페이지에서 클릭하면 현재 탭 정보로 어떤 행동을 수행한다. 보통은 저장소 API에 POST를 날리거나, 미리 작성한 입력 폼을 제목과 URL로 채운다. 구현은 두 방식으로 나뉜다.

직접 API로 보내는 방식은 매끄럽다. 로그인 토큰만 있으면 된다. 단점은 서비스별로 토큰 보관과 권한 설정을 신경 써야 한다는 점이다. 반대로 폼을 여는 방식은 안전하고 쉬운데, 마지막에 한 번 더 제출 버튼을 눌러야 한다.

다음은 폼을 열어 간단히 저장하는 예다. Notion을 쓰지 않는다면 자체 폼이나 Google Forms로 바꿔도 된다.

Javascript:(function() Trycatch(e) Alert('링크 저장 중 오류: ' + e.message);)();

API 방식의 예를 하나 더 보자. Raindrop.io에 저장하는 최소 구현이다. 토큰은 읽기 전용이 아닌 write 권한이어야 하며, 브라우저에 하드코딩할 때는 조직 내에서만 사용하는 것을 권한다.

Javascript:(async function() Const token = 'RAINDROP_TOKEN_HERE'; Const body = Link: location.href, Title: document.title ; Try Const res = await fetch('https://api.raindrop.io/rest/v1/raindrop', Method: 'POST', Headers: 'Authorization': 'Bearer ' + token, 'Content-Type': 'application/json' , Body: JSON.stringify(body)); If(!res.ok) throw new Error('HTTP ' + res.status) Alert('저장 완료'); catch(e) Alert('저장 실패: ' + e.message);)();

아래는 내가 북마클릿을 팀에 배포할 때 쓰는 요약 절차다. 한번의 설치로 모두가 같은 입력 경로를 쓰게 만드는 것이 목표다.

- 브라우저 즐겨찾기 바를 보이게 한다음, 자바스크립트 코드 스니펫을 새 북마크의 주소란에 붙인다.
- 저장소 API 토큰은 개인별로 발급해 각자 북마클릿에만 삽입한다. 공용 토큰을 쓰면 권한 추적이 흐려진다.
- 기본 태그를 북마클릿 코드에 하드코딩한다. 예를 들어 캠페인별 키워드, 프로젝트 코드 같은 것들이다.
- 모바일 대응을 위해 동일 동작을 하는 공유 시트 단축어를 만든다. iOS 단축어와 안드로이드 공유 인텐트로 처리한다.
- 설치 직후 테스트할 미션을 정해둔다. 예를 들어 오늘 본 기사 3개를 저장해 오도록 하고 피드에 노출되는지 확인한다.

태깅과 메타데이터가 검색력을 만든다

링크모음은 쌓이기만 하면 금방 쓸모를 잃는다. 검색을 견디려면 몇 가지 규칙이 필요하다. 첫째, 태그의 문법을 통일한다. 복수형과 단수형을 섞지 않고, 띄어쓰기 대신 하이픈이나 밑줄을 쓴다. 둘째, 소스 도메인을 별도로 적어 둔다. 나중에 도메인별 질을 평가하거나 차단 목록을 만들 때 요긴하다. 셋째, 요약 메모는 한두 문장으로 강제한다. 길어지면 제목과 중복되고, 짧으면 다시 열어봐야 한다.

중복 저장을 줄이는 방법도 있다. 저장 직전에 URL 정규화를 수행하는 것이다. 해시프래그먼트와 흔한 추적 파라미터를 제거하고, http를 https로 교체한다. 실무에서 많이 지우는 파라미터는 utm_source 계열, fbclid, gclid, ref 정도다. 플랫폼에 따라 필요한 경우도 있어 완전 제거 대신 화이트리스트를 먼저 만든다.

서비스 조합 예시

내가 추천하는 조합은 세 가지다. 완전 관리형, 혼합형, 셀프 호스팅이다. 각각 장단이 확실하다.

관리형은 Raindrop.io 같은 북마크 서비스 하나에 몰아 저장하고, 태그별 RSS를 바로 쓴다. 속도와 안정성이 좋고 모바일 경험이 훌륭하다. 협업 권한 관리도 쉽다. 단점은 고객 데이터가 외부에 있고 API 요금제 한도가 있다.

혼합형은 입력과 원본 저장은 관리형 서비스에 맡기되, 주기적으로 미러링해 정적 RSS와 JSON을 생성한다. GitHub Actions로 10분마다 동기화하고, 사이트와 뉴스레터는 미러 데이터를 쓴다. 외부 서비스 장애에도 배포가 계속 된다.

셀프 호스팅은 Supabase 같은 서버리스 백엔드에 테이블을 만들고, 북마클릿이 직접 API로 저장한다. Next.js나 Astro로 뷰를 만들고, RSS를 정적으로 생성한다. 완전한 제어권이 장점이지만, 보안과 유지보수의 문턱이 있다.

스포츠무료중계 키워드처럼 민감한 주제 다루기

링크 큐레이션을 하다 보면 트래픽이 높은 주제를 요청받는다. 스포츠무료중계 같은 검색어가 대표적이다. 여기에는 주의할 점이 많다. 불법 스트리밍 링크를 수집하거나 배포하면 저작권 문제에 걸릴 수 있다. 합법 서비스만 다룬다는 원칙을 세워야 한다. 공중파와 공식 OTT의 하이라이트, 리그가 직접 제공하는 무료 클립, 팀 공식 유튜브 채널 같은 것들만 포함한다. 사용자 기대를 관리하려면 큐레이션 페이지 상단에 안내를 둔다. 예를 들어 중계 정보는 지역 제한이 있을 수 있으며, 공식 채널 외 링크는 다루지 않는다는 문구다.

이 경우 태그 전략도 중요하다. Sports, league-kbo, official 같은 조합을 쓰면 RSS 구독자가 원하는 범위를 좁혀 받을 수 있다. 경기 일정은 링크가 아니라 캘린더로 해결하는 편이 낫다. 구글 캘린더 공개 주소를 소개하고, 경기별 공식 중계 링크만 이어 준다. 이렇게 하면 사이트 주소모음 성격과 합법성, 사용자 경험이 동시에 맞춰진다.

자동 검증과 품질 관리

수집이 늘어나면 깨진 링크와 리디렉션, 접근 제한이 늘어난다. 나는 매일 새벽에 링크 검사를 돌린다. 4xx와 5xx는 플래그를 세우고, 301과 302는 최종 URL로 업데이트한다. 서버가 과민하게 차단하는 경우를 피하려면 User-Agent를 조심스럽게 설정하고, 초당 요청을 제한한다.

간단한 파이썬 예시는 아래와 같다. 실제 운영에선 재시도와 백오프, 프록시 회전, 도메인별 규칙을 추가해야 한다.

Import requests, time From urllib.parse import urlparse, urlunparse Def normalize(url): U = requests.utils.urlparse(url) # 쿼리 파라미터 중 추적성 높은 것 제거 Q = '&'.join([kv for kv in u.query.split('&') if not kv.startswith(('utm_', 'gclid', 'fbclid', 'ref='))]) Return urlunparse((u.scheme or 'https', u.netloc, u.path, u.params, q, '')) Def check(url): Try: R = requests.head(url, allow_redirects=True, timeout=10, headers='User-Agent':'Mozilla/5.0 link-checker') Final = r.url Ok = r.status_code < 400 Return ok, final, r.status_code Except Exception as e: Return False, url, str(e) Urls = [...] # 오늘 추가된 링크 목록 For u in urls: N = normalize(u) Ok, final, code = check(n) Print(n, ok, final, code) Time.sleep(0.5)

검사 결과는 원본 저장소에 주석처럼 기록한다. 예를 들어 Raindrop.io에선 note 필드 뒤에 [alive:2026-05-12,status:200] 같은 꼬리표를 덧붙여 둔다. 뷰 레이어에서는 이 꼬리표를 읽어 죽은 링크에 취소선을 긋거나 대체 링크를 추천한다.

배포와 재활용

RSS로 나가는 순간부터 재활용은 쉬워진다. 태그별 RSS를 뉴스레터에 자동 삽입하고, 사이트에는 빌드 타임에 정적 HTML로 박아 넣는다. 나는 Astro를 써서 피드를 읽어 카드 형태로 렌더링한다. 카드에는 제목, 도메인, 요약, 태그 3개만 노출한다. 과하면 피곤하다. 모바일에선 2열 그리드를 1열로 바꿔 가독성을 챙긴다.

Slack과 Discord는 RSS 앱을 쓰면 바로 들어오지만, 메시지 포맷을 더 예쁘게 만들고 싶다면 중간에 작은 변환기를 둔다. RSS 항목을 받아 Open Graph 이미지를 미리 긁어오고, 도메인별 아이콘을 붙이는 정도만 해도 클릭률이 체감된다. 과거 실험에서 썸네일이 있는 링크는 없는 링크보다 20에서 30퍼센트 더 눌렸다.

Notion만 쓰는 팀이라면 임베드용 위젯을 하나 만든다. RSS를 받아서 반응형 카드로 렌더링하는 페이지를 Cloudflare Pages에 올리고, 그 URL을 노션에 임베드하면 끝이다. 업데이트는 RSS가 알아서 해 주니 노션은 항상 최신 상태를 비춘다.

팀 협업과 권한

여러 사람이 링크를 추가한다면 권한과 책임을 분리해야 한다. 수집 권한은 넓게, 배포 권한은 좁게 가져간다. 누군가가 잘못된 링크를 넣어도 곧장 외부로 나가지 않도록 검수 단계를 둔다. 방법은 간단하다. Tag:pending을 붙인 항목은 RSS에서 제외하고, 에디터가 tag:approved를 붙이면 다음 배포 주기에 포함시킨다. 댓글 기능이 있다면 이유를 짧게 남기게 한다. 무례함을 줄이는 데 의외로 큰 도움이 된다.

또 하나, 프로토콜이 필요하다. 링크 제목을 그대로 둘지, 편집자가 재작성할지, 요약은 어디까지 손댈지 정해 둔다. 시간이 지나면 이런 규칙이 차이를 만든다. 한때는 모든 요약을 직접 썼다가 과부하로 무너진 적이 있다. 이후로는 원문 핵심 구절을 인용하고 한 문장만 덧붙인다. 그 정도면 큐레이션의 견해도 살아 있고, 속도도 나온다.

유지보수 체크리스트

자동화라고 내버려두면 어느 순간 썩는다. 계정 만료, API 변경, 도메인 이전 같은 사건은 예고 없이 온다. 월별로 아래 항목만 돌려 보면 큰 사고를 막을 수 있다.

- 북마클릿 동작 확인, 토큰 만료일 점검, 팀 신규 입사자 온보딩 링크 최신화
- 태그 상위 50개 통계 점검, 유사 태그 병합, 금지 태그 목록 갱신
- 링크 검사 리포트 확인, 404 상위 도메인 블록리스트 후보 추출
- RSS 유효성 검증, 항목 수 과다 또는 과소 여부 확인, 구독자 피드백 수집
- 배포 경로별 클릭률 비교, 카드 템플릿 미세 조정, 이미지 크롤러 에러 로그 확인

문제 해결 사례

하나. 특정 언론사는 HEAD 요청을 차단해 링크 검사가 매일 실패했다. GET으로만 검사하면 트래픽이 너무 커졌고 차단 위험도 있었다. 해결은 단순했다. 그 도메인만 쿼리 파라미터 없앤 URL을 302 추적 없이 200으로 간주했다. 오탐이 조금 생겼지만 전체 안정성이 좋아졌다.

둘. 팀원이 모바일에서 자꾸 저장 실패를 겪었다. 확인해 보니 iOS에서 새 창 차단이 기본 설정이었다. 북마클릿을 새 창이 아닌 현재 탭에서 폼을 열도록 바꾸고, 제출 후 원래 페이지로 돌아오게 히스토리를 조작했다. 설치 가이드에 사파리 설정 스크린샷을 첨부한 것도 효과가 컸다.

셋. 링크가 하루에 300개씩 쌓이는 캠페인 기간이 있었다. 주간 뉴스레터를 자동 합본했더니 스팸처럼 보였다. 이후로는 배포 피드에 점수 필드를 추가했다. 저장 시 기본 1점, 에디터가 2점 이상으로 올린 항목만 뉴스레터에 포함되도록 룰을 바꿨다. 수치 기준을 주제별로 달리 설정해 품질이 올라갔다.

넷. 사이트 주소모음 페이지에서 도메인별 로고가 누락돼 보기에 성가셨다. Handshake나 신생 도메인은 파비콘을 구하기 어렵다. 로고 없을 때는 텍스트 배지로 대체하고, 도메인 첫 글자 색상 패턴을 정했다. 사용자는 통일감을 느끼고, 로고 수집 실패 로그도 눈에 잘 띄었다.

다섯. 외부 팀과 링크를 교환할 일이 생겼다. 서로 다른 태그 체계를 억지로 맞추려다 토론만 길어졌다. 결국 교환용 RSS를 따로 만들고, 맵핑 규칙을 변환기에 넣었다. 예를 들어 그쪽의 topic:ml은 우리 쪽의 tag:ai-ml로 변환한다. 내부 데이터는 건드리지 않고 협업만 매끄럽게 했다.

데이터 구조와 마이그레이션

도구를 오래 쓰다 보면 언젠가 옮길 일이 생긴다. 그래서 처음부터 필드를 과하게 서비스 종속적으로 설계하지 않는다. 최소한의 공통 필드는 title, url, tags, note, createdAt, sourceDomain 정도다. 서비스가 제공하는 추가 필드는 별도의 namespace로 note 끝에 넣거나, 커스텀 JSON 필드에 보관한다.

마이그레이션은 내보내기와 들여오기 사이의 간극을 메우는 일이다. 예를 들어 Pinboard에서 Raindrop.io로 옮길 때 북마크의 star 상태를 즐겨찾기 컬렉션으로 치환하고, private 플래그는 태그로 보존한다. RSS로만 들여오는 서비스라면 1회성으로 10만 건을 밀어 넣기 어렵다. 이럴 땐 JSON 백업을 S3에 올리고, 조회 레이어에서 둘을 병합한 뒤 90일에 걸쳐 점진 이관한다. 급하게 완전 이주를 시도하면 링크 품질이 무너진다.

보안과 개인정보

북마클릿에 토큰을 넣는 구조는 편하지만 위험하다. 현실적인 타협은 네 가지다. 첫째, 토큰 범위를 최소화한다. Write 권한만 주고, read와 admin은 빼 둔다. 둘째, 도메인 제한이 가능한 토큰을 쓰거나, 프록시 API를 만들어 CORS 기준을 우리 도메인으로만 묶는다. 셋째, 토큰 교체 주기를 정해 분기마다 폐기한다. 넷째, 개인 북마클릿과 팀 북마클릿을 명확히 구분하고, 유출 사고가 나면 즉시 관련 토큰만 폐기한다.

개인정보는 더 조심해야 한다. 내부 티켓이나 비공개 문서 링크가 외부 피드에 섞여 나가는 사고가 종종 발생한다. 기본값을 private로 하고, 공개 태그를 붙여야만 외부 RSS에 실리도록 룰을 만든다. 내부 피드와 외부 피드의 엔드포인트를 완전히 분리하고 접근 로그를 남겨 둔다.

작은 자동화가 만드는 큰 차이

자동화는 거창한 로봇이 아니라, 몇 초를 줄이는 작은 습관의 누적이다. 브라우저에서 클릭 한 번이면 저장이 끝나고, 태그는 제안으로 채워지며, RSS는 알아서 흘러간다. 나는 링크에 쓰는 시간을 거의 절반으로 줄였고, 주간 링크 모음 발행 실패가 사실상 사라졌다. 무엇보다 팀원이 새로운 주제를 시도할 때 발목을 잡지 않게 됐다. 사이트 주소 모음 페이지에 새 섹션을 여는 일도, 링크모음 뉴스레터에 새로운 카테고리를 도입하는 일도, 이제는 부담이 아니다.

핵심은 표준과 단순함이다. RSS는 오래 살아남은 표준이고, 북마클릿은 브라우저가 있는 한 계속 쓸 수 있다. 나머지는 취향과 상황에 맞춰 붙였다 떼면 된다. 민감한 주제, 예컨대 스포츠무료중계 같은 요청이 들어와도 원칙만 지키면 흔들리지 않는다. 합법 소스만 다룬다는 규칙, 중복과 쓰레기를 줄이는 태그 전략, 죽은 링크를 부지런히 걷어내는 점검 루틴이 시스템의 척추가 된다.

복잡한 도구보다 중요한 것은 흐름을 한 번 설계하는 일이다. 수집 - 정리 - 배포. 이 세 단계를 잇는 끈으로 RSS와 북마클릿을 고른다면, 내게 필요한 링크 시스템은 어렵지 않게 완성된다. 그리고 그 시스템은 시간이 갈수록 더 강해진다.