



Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'TAILQ_INSERT_AFTER.3'

\$ man TAILQ_INSERT_AFTER.3

TAILQ(3) Library Functions Manual TAILQ(3)

NAME

TAILQ_CONCAT, TAILQ_EMPTY, TAILQ_ENTRY, TAILQ_FIRST, TAILQ_FOREACH, TAILQ_FOREACH_REVERSE, TAILQ_HEAD, TAILQ_HEAD_INITIALIZER, TAILQ_INIT, TAILQ_INSERT_AFTER, TAILQ_INSERT_BEFORE, TAILQ_INSERT_HEAD, TAILQ_INSERT_TAIL, TAILQ_LAST, TAILQ_NEXT, TAILQ_PREV, TAILQ_REMOVE - implementation of a doubly linked tail queue

LIBRARY

Standard C library (libc, -lc)

SYNOPSIS

```
#include <sys/queue.h>

TAILQ_ENTRY(TYPE);

TAILQ_HEAD(HEADNAME, TYPE);

TAILQ_HEAD TAILQ_HEAD_INITIALIZER(TAILQ_HEAD head);

void TAILQ_INIT(TAILQ_HEAD *head);

int TAILQ_EMPTY(TAILQ_HEAD *head);

void TAILQ_INSERT_HEAD(TAILQ_HEAD *head,
    struct TYPE *elm, TAILQ_ENTRY NAME);

void TAILQ_INSERT_TAIL(TAILQ_HEAD *head,
    struct TYPE *elm, TAILQ_ENTRY NAME);

void TAILQ_INSERT_BEFORE(struct TYPE *listelm,
    struct TYPE *elm, TAILQ_ENTRY NAME);

void TAILQ_INSERT_AFTER(TAILQ_HEAD *head, struct TYPE *listelm,
```

```

        struct TYPE *elm, TAILQ_ENTRY NAME);

struct TYPE *TAILQ_FIRST(TAILQ_HEAD *head);

struct TYPE *TAILQ_LAST(TAILQ_HEAD *head, HEADNAME);

struct TYPE *TAILQ_PREV(struct TYPE *elm, HEADNAME, TAILQ_ENTRY NAME);

struct TYPE *TAILQ_NEXT(struct TYPE *elm, TAILQ_ENTRY NAME);

TAILQ_FOREACH(struct TYPE *var, TAILQ_HEAD *head,

        TAILQ_ENTRY NAME);

TAILQ_FOREACH_REVERSE(struct TYPE *var, TAILQ_HEAD *head, HEADNAME,

        TAILQ_ENTRY NAME);

void TAILQ_REMOVE(TAILQ_HEAD *head, struct TYPE *elm,

        TAILQ_ENTRY NAME);

void TAILQ_CONCAT(TAILQ_HEAD *head1, TAILQ_HEAD *head2,

        TAILQ_ENTRY NAME);

```

DESCRIPTION

These macros define and operate on doubly linked tail queues.

In the macro definitions, TYPE is the name of a user defined structure, that must contain a field of type TAILQ_ENTRY, named NAME. The argument HEADNAME is the name of a user defined structure that must be declared using the macro TAILQ_HEAD().

Creation

A tail queue is headed by a structure defined by the TAILQ_HEAD() macro. This structure contains a pair of pointers, one to the first element in the queue and the other to the last element in the queue. The elements are doubly linked so that an arbitrary element can be removed without traversing the queue. New elements can be added to the queue after an existing element, before an existing element, at the head of the queue, or at the end of the queue. A TAILQ_HEAD structure is declared as follows:

```
TAILQ_HEAD(HEADNAME, TYPE) head;
```

where struct HEADNAME is the structure to be defined, and struct TYPE is the type of the elements to be linked into the queue. A pointer to the head of the queue can later be declared as:

```
struct HEADNAME *headp;
```

(The names head and headp are user selectable.)

TAILQ_ENTRY() declares a structure that connects the elements in the queue.

TAILQ_HEAD_INITIALIZER() evaluates to an initializer for the queue head.

TAILQ_INIT() initializes the queue referenced by

TAILQ_EMPTY() evaluates to true if there are no items on the queue. head.

Insertion

TAILQ_INSERT_HEAD() inserts the new element elm at the head of the queue.

TAILQ_INSERT_TAIL() inserts the new element elm at the end of the queue.

TAILQ_INSERT_BEFORE() inserts the new element elm before the element listelm.

TAILQ_INSERT_AFTER() inserts the new element elm after the element listelm.

Traversal

TAILQ_FIRST() returns the first item on the queue, or NULL if the queue is empty.

TAILQ_LAST() returns the last item on the queue. If the queue is empty the return value is NULL.

TAILQ_PREV() returns the previous item on the queue, or NULL if this item is the first.

TAILQ_NEXT() returns the next item on the queue, or NULL if this item is the last.

TAILQ_FOREACH() traverses the queue referenced by head in the forward direction, assigning each element in turn to var. var is set to NULL if the loop completes normally, or if there were no elements.

TAILQ_FOREACH_REVERSE() traverses the queue referenced by head in the reverse direction, assigning each element in turn to var.

Removal

TAILQ_REMOVE() removes the element elm from the queue.

Other features

TAILQ_CONCAT() concatenates the queue headed by head2 onto the end of the one headed by head1 removing all entries from the former.

RETURN VALUE

TAILQ_EMPTY() returns nonzero if the queue is empty, and zero if the queue contains at least one entry.

TAILQ_FIRST(), TAILQ_LAST(), TAILQ_PREV(), and TAILQ_NEXT() return a pointer to the first, last, previous, or next TYPE structure, respectively.

TAILQ_HEAD_INITIALIZER() returns an initializer that can be assigned to the queue head.

STANDARDS

BSD.

HISTORY

4.4BSD.

CAVEATS

TAILQ_FOREACH() and TAILQ_FOREACH_REVERSE() don't allow var to be removed or freed within the loop, as it would interfere with the traversal. TAILQ_FOREACH_SAFE() and TAILQ_FOREACH_REVERSE_SAFE(), which are present on the BSDs but are not present in glibc, fix this limitation by allowing var to safely be removed from the list and freed from within the loop without interfering with the traversal.

EXAMPLES

```
#include <stddef.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/queue.h>

struct entry {
    int data;

    TAILQ_ENTRY(entry) entries;      /* Tail queue */
};

TAILQ_HEAD(tailhead, entry);

int
main(void)
{
    struct entry *n1, *n2, *n3, *np;
    struct tailhead head;           /* Tail queue head */
    int i;

    TAILQ_INIT(&head);              /* Initialize the queue */
    n1 = malloc(sizeof(struct entry)); /* Insert at the head */
    TAILQ_INSERT_HEAD(&head, n1, entries);
    n1 = malloc(sizeof(struct entry)); /* Insert at the tail */
    TAILQ_INSERT_TAIL(&head, n1, entries);
    n2 = malloc(sizeof(struct entry)); /* Insert after */
    TAILQ_INSERT_AFTER(&head, n1, n2, entries);
    n3 = malloc(sizeof(struct entry)); /* Insert before */
    TAILQ_INSERT_BEFORE(n2, n3, entries);
    TAILQ_REMOVE(&head, n2, entries); /* Deletion */
    free(n2);
```

```

        /* Forward traversal */

i = 0;

TAILQ_FOREACH(np, &head, entries)
    np->data = i++;

        /* Reverse traversal */

TAILQ_FOREACH_REVERSE(np, &head, tailhead, entries)
    printf("%i\n", np->data);

        /* TailQ deletion */

n1 = TAILQ_FIRST(&head);
while (n1 != NULL) {
    n2 = TAILQ_NEXT(n1, entries);
    free(n1);
    n1 = n2;
}

TAILQ_INIT(&head);
exit(EXIT_SUCCESS);
}

```

SEE ALSO

insque(3), queue(7)

Linux man-pages 6.17

2025-05-17

TAILQ(3)