Gnus Manual

by Lars Magne Ingebrigtsen

The Gnus Newsreader

Gnus is the advanced, self-documenting, customizable, extensible unreal-time newsreader for GNU Emacs.

Oops. That sounds oddly familiar, so let's start over again to avoid being accused of plagiarism:

Gnus is a message-reading laboratory. It will let you look at just about anything as if it were a newsgroup. You can read mail with it, you can browse directories with it, you can `ftp` with it—you can even read news with it!

Gnus tries to empower people who read news the same way Emacs empowers people who edit text. Gnus sets no limits to what the user should be allowed to do. Users are encouraged to extend Gnus to make it behave like they want it to behave. A program should not control people; people should be empowered to do what they want by using (or abusing) the program.

1 Starting Gnus

If your system administrator has set things up properly, starting Gnus and reading news is extremely easy—you just type *M-x gnus* in your Emacs.

If you want to start Gnus in a different frame, you can use the command *M-x gnus-other-frame* instead.

If things do not go smoothly at startup, you have to twiddle some variables.

1.1 Finding the News

The `gnus-select-method` variable says where Gnus should look for news. This variable should be a list where the first element says *how* and the second element says *where*. This method is your native method. All groups that are not fetched with this method are foreign groups.

For instance, if the '`news.somewhere.edu`' NNTP server is where you want to get your daily dosage of news from, you'd say:

```
(setq gnus-select-method '(nntp "news.somewhere.edu"))
```

If you want to read directly from the local spool, say:

```
(setq gnus-select-method '(nnspool ""))
```

If you can use a local spool, you probably should, as it will almost certainly be much faster.

If this variable is not set, Gnus will take a look at the `NNTPSERVER` environment variable. If that variable isn't set, Gnus will see whether `gnus-nntpserver-file` ('`/etc/nntpserver`' by default) has any opinions on the matter. If that fails as well, Gnus will will try to use the machine that is running Emacs as an NNTP server. That's a long-shot, though.

If `gnus-nntp-server` is set, this variable will override `gnus-select-method`. You should therefore set `gnus-nntp-server` to `nil`, which is what it is by default.

You can also make Gnus prompt you interactively for the name of an NNTP server. If you give a non-numerical prefix to `gnus` (i.e., *C-u M-x gnus*), Gnus will let you choose between the servers in the `gnus-secondary-servers` list (if any). You can also just type in the name of any server you feel like visiting.

However, if you use one NNTP server regularly and are just interested in a couple of groups from a different server, you would be better served by using the *B* command in the group buffer. It will let you have a look at what groups are available, and you can subscribe to any of the groups you want to. This also makes '`.newsrc`' maintenance much tidier. See Section 2.8 [Foreign Groups], page 18.

A slightly different approach to foreign groups is to set the `gnus-secondary-select-methods` variable. The select methods listed in this variable are in many ways just as native as the `gnus-select-method` server. They will also be queried for active files during startup (if that's required), and new newsgroups that appear on these servers will be subscribed (or not) just as native groups are.

For instance, if you use the `nnmbox` backend to read your mail, you would typically set this variable to

```
(setq gnus-secondary-select-methods '((nnmbox "")))
```

1.2 The First Time

If no startup files exist, Gnus will try to determine what groups should be subscribed by default.

If the variable `gnus-default-subscribed-newsgroups` is set, Gnus will subscribe you to just those groups in that list, leaving the rest killed. Your system administrator should have set this variable to something useful.

Since she hasn't, Gnus will just subscribe you to a few arbitrarily picked groups (i.e., '`*.newusers`'). (*Arbitrary* is here defined as *whatever Lars thinks you should read.*)

You'll also be subscribed to the Gnus documentation group, which should help you with most common problems.

If `gnus-default-subscribed-newsgroups` is `t`, Gnus will just use the normal functions for handling new groups, and not do anything special.

1.3 The Server is Down

If the default server is down, Gnus will understandably have some problems starting. However, if you have some mail groups in addition to the news groups, you may want to start Gnus anyway.

Gnus, being the trusting sort of program, will ask whether to proceed without a native select method if that server can't be contacted. This will happen whether the server doesn't actually exist (i.e., you have given the wrong address) or the server has just momentarily taken ill for some reason or other. If you decide to continue and have no foreign groups, you'll find it difficult to actually do anything in the group buffer. But, hey, that's your problem. Blllrph!

If you know that the server is definitely down, or you just want to read your mail without bothering with the server at all, you can use the `gnus-no-server` command to start Gnus. That might come in handy if you're in a hurry as well.

1.4 Slave Gnusi

You might want to run more than one Emacs with more than one Gnus at the same time. If you are using different '`.newsrc`' files (eg., if you are using the two different Gnusi to read from two different servers), that is no problem whatsoever. You just do it.

The problem appears when you want to run two Gnusi that use the same `.newsrc` file.

To work around that problem some, we here at the Think-Tank at the Gnus Towers have come up with a new concept: *Masters* and *servants*. (We have applied for a patent

on this concept, and have taken out a copyright on those words. If you wish to use those words in conjunction with each other, you have to send $1 per usage instance to me. Usage of the patent (*Master/Slave Relationships In Computer Applications*) will be much more expensive, of course.)

Anyways, you start one Gnus up the normal way with *M-x gnus* (or however you do it). Each subsequent slave Gnusi should be started with *M-x gnus-slave*. These slaves won't save normal '`.newsrc`' files, but instead save *slave files* that contains information only on what groups have been read in the slave session. When a master Gnus starts, it will read (and delete) these slave files, incorporating all information from them. (The slave files will be read in the sequence they were created, so the latest changes will have precedence.)

Information from the slave files has, of course, precedence over the information in the normal (i. e., master) `.newsrc` file.

1.5 Fetching a Group

It it sometime convenient to be able to just say "I want to read this group and I don't care whether Gnus has been started or not". This is perhaps more useful for people who write code than for users, but the command `gnus-fetch-group` provides this functionality in any case. It takes the group name as a parameter.

1.6 New Groups

What Gnus does when it encounters a new group is determined by the `gnus-subscribe-newsgroup-method` variable.

This variable should contain a function. Some handy pre-fab values are:

`gnus-subscribe-zombies`
: Make all new groups zombies. You can browse the zombies later (with *A z*) and either kill them all off properly, or subscribe to them. This is the default.

`gnus-subscribe-randomly`
: Subscribe all new groups randomly.

`gnus-subscribe-alphabetically`
: Subscribe all new groups alphabetically.

`gnus-subscribe-hierarchically`
: Subscribe all new groups hierarchically.

`gnus-subscribe-interactively`
: Subscribe new groups interactively. This means that Gnus will ask you about **all** new groups.

`gnus-subscribe-killed`
: Kill all new groups.

A closely related variable is `gnus-subscribe-hierarchical-interactive`. (That's quite a mouthful.) If this variable is non-`nil`, Gnus will ask you in a hierarchical fashion whether to subscribe to new groups or not. Gnus will ask you for each sub-hierarchy whether you want to descend the hierarchy or not.

One common mistake is to set the variable a few paragraphs above to `gnus-subscribe-hierarchical-interactive`. This is an error. This will not work. This is ga-ga. So don't do it.

A nice and portable way to control which new newsgroups should be subscribed (or ignored) is to put an *options* line at the start of the '`.newsrc`' file. Here's an example:

```
options -n !alt.all !rec.all sci.all
```

This line obviously belongs to a serious-minded intellectual scientific person (or she may just be plain old boring), because it says that all groups that have names beginning with '`alt`' and '`rec`' should be ignored, and all groups with names beginning with '`sci`' should be subscribed. Gnus will not use the normal subscription method for subscribing these groups. `gnus-subscribe-options-newsgroup-method` is used instead. This variable defaults to `gnus-subscribe-alphabetically`.

If you don't want to mess with your '`.newsrc`' file, you can just set the two variables `gnus-options-subscribe` and `gnus-options-not-subscribe`. These two variables do exactly the same as the '`.newsrc`' '`options -n`' trick. Both are regexps, and if the the new group matches the former, it will be unconditionally subscribed, and if it matches the latter, it will be ignored.

Yet another variable that meddles here is `gnus-auto-subscribed-groups`. It works exactly like `gnus-options-subscribe`, and is therefore really superfluous, but I thought it would be nice to have two of these. This variable is more meant for setting some ground rules, while the other variable is used more for user fiddling. By default this variable makes all new groups that come from mail backends (`nnml`, `nnbabyl`, `nnfolder`, `nnmbox`, and `nnmh`) subscribed. If you don't like that, just set this variable to `nil`.

If you are satisfied that you really never want to see any new groups, you could set `gnus-check-new-newsgroups` to `nil`. This will also save you some time at startup. Even if this variable is `nil`, you can always subscribe to the new groups just by pressing *U* in the group buffer (see Section 2.12 [Group Maintenance], page 22). This variable is `t` by default.

Gnus normally determines whether a group is new or not by comparing the list of groups from the active file(s) with the lists of subscribed and dead groups. This isn't a particularly fast method. If `gnus-check-new-newsgroups` is `ask-server`, Gnus will ask the server for new groups since the last time. This is both faster & cheaper. This also means that you can get rid of the list of killed groups altogether, so you may set `gnus-save-killed-list` to `nil`, which will save time both at startup, at exit, and all over. Saves disk space, too. Why isn't this the default, then? Unfortunately, not all servers support this command.

I bet I know what you're thinking now: How do I find out whether my server supports `ask-server`? No? Good, because I don't have a fail-safe answer. I would suggest just setting this variable to `ask-server` and see whether any new groups appear within the next few days. If any do, then it works. If any don't, then it doesn't work. I could write a function to make Gnus guess whether the server supports `ask-server`, but it would just be a guess. So I won't. You could `telnet` to the server and say `HELP` and see whether it

lists '`NEWGROUPS`' among the commands it understands. If it does, then it might work. (But there are servers that lists '`NEWGROUPS`' without supporting the function properly.)

This variable can also be a list of select methods. If so, Gnus will issue an `ask-server` command to each of the select methods, and subscribe them (or not) using the normal methods. This might be handy if you are monitoring a few servers for new groups. A side effect is that startup will take much longer, so you can meditate while waiting. Use the mantra "dingnusdingnusdingnus" to achieve permanent bliss.

1.7 Startup Files

Now, you all know about the '`.newsrc`' file. All subscription information is traditionally stored in this file.

Things got a bit more complicated with GNUS. In addition to keeping the '`.newsrc`' file updated, it also used a file called '`.newsrc.el`' for storing all the information that didn't fit into the '`.newsrc`' file. (Actually, it also duplicated everything in the '`.newsrc`' file.) GNUS would read whichever one of these files was the most recently saved, which enabled people to swap between GNUS and other newsreaders.

That was kinda silly, so Gnus went one better: In addition to the '`.newsrc`' and '`.newsrc.el`' files, Gnus also has a file called '`.newsrc.eld`'. It will read whichever of these files that are most recent, but it will never write a '`.newsrc.el`' file.

You can turn off writing the '`.newsrc`' file by setting `gnus-save-newsrc-file` to `nil`, which means you can delete the file and save some space, as well as making exit from Gnus faster. However, this will make it impossible to use other newsreaders than Gnus. But hey, who would want to, right?

If `gnus-save-killed-list` (default `t`) is `nil`, Gnus will not save the list of killed groups to the startup file. This will save both time (when starting and quitting) and space (on disk). It will also means that Gnus has no record of what groups are new or old, so the automatic new groups subscription methods become meaningless. You should always set `gnus-check-new-newsgroups` to `nil` or `ask-server` if you set this variable to `nil` (see Section 1.6 [New Groups], page 5).

The `gnus-startup-file` variable says where the startup files are. The default value is '`~/.newsrc`', with the Gnus (El Dingo) startup file being whatever that one is with a '`.eld`' appended.

`gnus-save-newsrc-hook` is called before saving any of the newsrc files, while `gnus-save-quick-newsrc-hook` is called just before saving the '`.newsrc.eld`' file, and `gnus-save-standard-newsrc-hook` is called just before saving the '`.newsrc`' file. The latter two are commonly used to turn version control on or off. Version control is off by default when saving the startup files.

1.8 Auto Save

Whenever you do something that changes the Gnus data (reading articles, catching up, killing/subscribing groups), the change is added to a special *dribble buffer*. This buffer is

auto-saved the normal Emacs way. If your Emacs should crash before you have saved the '`.newsrc`' files, all changes you have made can be recovered from this file.

If Gnus detects this file at startup, it will ask the user whether to read it. The auto save file is deleted whenever the real startup file is saved.

If `gnus-use-dribble-file` is `nil`, Gnus won't create and maintain a dribble buffer. The default is `t`.

Gnus will put the dribble file(s) in `gnus-dribble-directory`. If this variable is `nil`, which it is by default, Gnus will dribble into the directory where the '`.newsrc`' file is located. (This is normally the user's home directory.) The dribble file will get the same file permissions as the `.newsrc` file.

1.9 The Active File

When Gnus starts, or indeed whenever it tries to determine whether new articles have arrived, it reads the active file. This is a very large file that lists all the active groups and articles on the server.

Before examining the active file, Gnus deletes all lines that match the regexp `gnus-ignored-newsgroups`. This is done primarily to reject any groups with bogus names, but you can use this variable to make Gnus ignore hierarchies you aren't ever interested in. However, this is not recommended. In fact, it's highly discouraged. Instead, see Section 1.6 [New Groups], page 5 for an overview of other variables that can be used instead.

The active file can be rather Huge, so if you have a slow network, you can set `gnus-read-active-file` to `nil` to prevent Gnus from reading the active file. This variable is `t` by default.

Gnus will try to make do by getting information just on the groups that you actually subscribe to.

Note that if you subscribe to lots and lots of groups, setting this variable to `nil` will probably make Gnus slower, not faster. At present, having this variable `nil` will slow Gnus down considerably, unless you read news over a 2400 baud modem.

This variable can also have the value `some`. Gnus will then attempt to read active info only on the subscribed groups. On some servers this is quite fast (on sparkling, brand new INN servers that support the `LIST ACTIVE group` command), on others this isn't fast at all. In any case, `some` should be faster than `nil`, and is certainly faster than `t` over slow lines.

If this variable is `nil`, Gnus will ask for group info in total lock-step, which isn't very fast. If it is `some` and you use an NNTP server, Gnus will pump out commands as fast as it can, and read all the replies in one swoop. This will normally result in better performance, but if the server does not support the aforementioned `LIST ACTIVE group` command, this isn't very nice to the server.

In any case, if you use `some` or `nil`, you should definitely kill all groups that you aren't interested in to speed things up.

1.10 Startup Variables

`gnus-load-hook`
: A hook that is run while Gnus is being loaded. Note that this hook will normally be run just once in each Emacs session, no matter how many times you start Gnus.

`gnus-startup-hook`
: A hook that is run after starting up Gnus successfully.

`gnus-check-bogus-newsgroups`
: If non-`nil`, Gnus will check for and delete all bogus groups at startup. A *bogus group* is a group that you have in your '`.newsrc`' file, but doesn't exist on the news server. Checking for bogus groups can take quite a while, so to save time and resources it's best to leave this option off, and do the checking for bogus groups once in a while from the group buffer instead (see Section 2.12 [Group Maintenance], page 22).

`gnus-inhibit-startup-message`
: If non-`nil`, the startup message won't be displayed. That way, your boss might not notice that you are reading news instead of doing your job as easily.

`gnus-no-groups-message`
: Message displayed by Gnus when no groups are available.

2 The Group Buffer

The *group buffer* lists all (or parts) of the available groups. It is the first buffer shown when Gnus starts, and will never be killed as long as Gnus is active.

2.1 Group Buffer Format

2.1.1 Group Line Specification

The default format of the group buffer is nice and dull, but you can make it as exciting and ugly as you feel like.

Here's a couple of example group lines:

```
      25: news.announce.newusers
*      0: alt.fan.andrea-dworkin
```

Quite simple, huh?

You can see that there are 25 unread articles in '`news.announce.newusers`'. There are no unread articles, but some ticked articles, in '`alt.fan.andrea-dworkin`' (see that little asterisk at the beginning of the line?)

You can change that format to whatever you want by fiddling with the `gnus-group-line-format` variable. This variable works along the lines of a `format` specification, which is pretty much the same as a `printf` specifications, for those of you who use (feh!) C. See Section 8.3 [Formatting Variables], page 120.

The default value that produced those lines above is '`%M%S%5y: %(%g%)\n`'.

There should always be a colon on the line; the cursor always moves to the colon after performing an operation. Nothing else is required—not even the group name. All displayed text is just window dressing, and is never examined by Gnus. Gnus stores all real information it needs using text properties.

(Note that if you make a really strange, wonderful, spreadsheet-like layout, everybody will believe you are hard at work with the accounting instead of wasting time reading news.)

Here's a list of all available format characters:

'`M`' Only marked articles.

'`S`' Whether the group is subscribed.

'`L`' Level of subscribedness.

'`N`' Number of unread articles.

'`I`' Number of dormant articles.

'`T`' Number of ticked articles.

'`R`' Number of read articles.

‘t’ Total number of articles.

‘y’ Number of unread, unticked, non-dormant articles.

‘i’ Number of ticked and dormant articles.

‘g’ Full group name.

‘G’ Group name.

‘D’ Newsgroup description.

‘o’ ‘m’ if moderated.

‘O’ ‘(m)’ if moderated.

‘s’ Select method.

‘n’ Select from where.

‘z’ A string that looks like ‘<%s:%n>’ if a foreign select method is used.

‘P’ Indentation based on the level of the topic (see Section 2.15 [Group Topics], page 24).

‘c’ Short (collapsed) group name. The `gnus-group-uncollapsed-levels` variable says how many levels to leave at the end of the group name. The default is `1`.

‘u’ User defined specifier. The next character in the format string should be a letter. GNUS will call the function `gnus-user-format-function-`‘X’, where ‘X’ is the letter following ‘%u’. The function will be passed the current headers as argument. The function should return a string, which will be inserted into the buffer just like information from any other specifier.

All the “number-of” specs will be filled with an asterisk (‘*’) if no info is available—for instance, if it is a non-activated foreign group, or a bogus (or semi-bogus) native group.

2.1.2 Group Modeline Specification

The mode line can be changed by setting (`gnus-group-mode-line-format`). It doesn't understand that many format specifiers:

‘S’ The native news server.

‘M’ The native select method.

2.1.3 Group Highlighting

Highlighting in the group buffer is controlled by the `gnus-group-highlight` variable. This is an alist with elements that look like *(form . face)*. If *form* evaluates to something non-`nil`, the *face* will be used on the line.

Here's an example value for this variable that might look nice if the background is dark:

```
(setq gnus-group-highlight
    `(((> unread 200) .
       ,(custom-face-lookup "Red" nil nil t nil nil))
      ((and (< level 3) (zerop unread)) .
       ,(custom-face-lookup "SeaGreen" nil nil t nil nil))
      ((< level 3) .
       ,(custom-face-lookup "SpringGreen" nil nil t nil nil))
      ((zerop unread) .
       ,(custom-face-lookup "SteelBlue" nil nil t nil nil))
      (t .
       ,(custom-face-lookup "SkyBlue" nil nil t nil nil))
      ))
```

Variables that are dynamically bound when the forms are evaluated include:

`group` The group name.

`unread` The number of unread articles in the group.

`method` The select method.

`mailp` Whether the group is a mail group.

`level` The level of the group.

`score` The score of the group.

`ticked` The number of ticked articles in the group.

`topic` When using the topic minor mode, this variable is bound to the current topic being inserted.

When the forms are `eval`ed, point is at the beginning of the line of the group in question, so you can use many of the normal Gnus functions for snarfing info on the group.

`gnus-group-update-hook` is called when a group line is changed. It will not be called when `gnus-visual` is `nil`. This hook calls `gnus-group-highlight-line` by default.

2.2 Group Maneuvering

All movement commands understand the numeric prefix and will behave as expected, hopefully.

n Go to the next group that has unread articles (`gnus-group-next-unread-group`).

p
DEL Go to the previous group group that has unread articles (`gnus-group-prev-unread-group`).

N Go to the next group (`gnus-group-next-group`).

P Go to the previous group (`gnus-group-prev-group`).

M-p Go to the next unread group on the same level (or lower) (`gnus-group-next-unread-group-same-level`).

M-n Go to the previous unread group on the same level (or lower) (`gnus-group-prev-unread-group-same-level`).

Three commands for jumping to groups:

j Jump to a group (and make it visible if it isn't already) (`gnus-group-jump-to-group`). Killed groups can be jumped to, just like living groups.

, Jump to the unread group with the lowest level (`gnus-group-best-unread-group`).

. Jump to the first group with unread articles (`gnus-group-first-unread-group`).

If `gnus-group-goto-unread` is `nil`, all the movement commands will move to the next group, not the next unread group. Even the commands that say they move to the next unread group. The default is `t`.

2.3 Selecting a Group

SPACE Select the current group, switch to the summary buffer and display the first unread article (`gnus-group-read-group`). If there are no unread articles in the group, or if you give a non-numerical prefix to this command, Gnus will offer to fetch all the old articles in this group from the server. If you give a numerical prefix *N*, Gnus will fetch *N* number of articles. If *N* is positive, fetch the *N* newest articles, if *N* is negative, fetch the *abs(N)* oldest articles.

RET Select the current group and switch to the summary buffer (`gnus-group-select-group`). Takes the same arguments as `gnus-group-read-group`—the only difference is that this command does not display the first unread article automatically upon group entry.

M-RET This does the same as the command above, but tries to do it with the minimum amount off fuzz (`gnus-group-quick-select-group`). No scoring/killing will be performed, there will be no highlights and no expunging. This might be useful if you're in a real hurry and have to enter some humongous group.

M-SPACE This is yet one more command that does the same as the one above, but this one does it without expunging and hiding dormants (`gnus-group-visible-select-group`).

c Mark all unticked articles in this group as read (`gnus-group-catchup-current`). `gnus-group-catchup-group-hook` is when catching up a group from the group buffer.

C Mark all articles in this group, even the ticked ones, as read (`gnus-group-catchup-current-all`).

The `gnus-large-newsgroup` variable says what Gnus should consider to be a big group. This is 200 by default. If the group has more unread articles than this, Gnus will query the user before entering the group. The user can then specify how many articles should be

fetched from the server. If the user specifies a negative number (`-n`), the `n` oldest articles will be fetched. If it is positive, the `n` articles that have arrived most recently will be fetched.

`gnus-auto-select-first` control whether any articles are selected automatically when entering a group.

`nil` Don't select any articles when entering the group. Just display the full summary buffer.

`t` Select the first unread article when entering the group.

`best` Select the most high-scored article in the group when entering the group.

If you want to prevent automatic selection in some group (say, in a binary group with Huge articles) you can set this variable to `nil` in `gnus-select-group-hook`, which is called when a group is selected.

2.4 Subscription Commands

S t
u Toggle subscription to the current group (`gnus-group-unsubscribe-current-group`).

S s
U Prompt for a group to subscribe, and then subscribe it. If it was subscribed already, unsubscribe it instead (`gnus-group-unsubscribe-group`).

S k
C-k Kill the current group (`gnus-group-kill-group`).

S y
C-y Yank the last killed group (`gnus-group-yank-group`).

C-x C-t Transpose two groups (`gnus-group-transpose-groups`). This isn't really a subscription command, but you can use it instead of a kill-and-yank sequence sometimes.

S w
C-w Kill all groups in the region (`gnus-group-kill-region`).

S z Kill all zombie groups (`gnus-group-kill-all-zombies`).

S C-k Kill all groups on a certain level (`gnus-group-kill-level`). These groups can't be yanked back after killing, so this command should be used with some caution. The only thing where this command comes in really handy is when you have a '`.newsrc`' with lots of unsubscribed groups that you want to get rid off. *S C-k* on level `7` will kill off all unsubscribed groups that do not have message numbers in the '`.newsrc`' file.

Also see Section 2.5 [Group Levels], page 16.

2.5 Group Levels

All groups have a level of *subscribedness*. For instance, if a group is on level 2, it is more subscribed than a group on level 5. You can ask Gnus to just list groups on a given level or lower (see Section 2.10 [Listing Groups], page 21), or to just check for new articles in groups on a given level or lower (see Section 2.16.1 [Scanning New Messages], page 27).

Remember: The higher the level of the group, the less important it is.

`S l` Set the level of the current group. If a numeric prefix is given, the next *n* groups will have their levels set. The user will be prompted for a level.

Gnus considers groups on between levels 1 and `gnus-level-subscribed` (inclusive) (default 5) to be subscribed, `gnus-level-subscribed` (exclusive) and `gnus-level-unsubscribed` (inclusive) (default 7) to be unsubscribed, `gnus-level-zombie` to be zombies (walking dead) (default 8) and `gnus-level-killed` to be killed (default 9), completely dead. Gnus treats subscribed and unsubscribed groups exactly the same, but zombie and killed groups have no information on what articles you have read, etc, stored. This distinction between dead and living groups isn't done because it is nice or clever, it is done purely for reasons of efficiency.

It is recommended that you keep all your mail groups (if any) on quite low levels (eg. 1 or 2).

If you want to play with the level variables, you should show some care. Set them once, and don't touch them ever again. Better yet, don't touch them at all unless you know exactly what you're doing.

Two closely related variables are `gnus-level-default-subscribed` (default 3) and `gnus-level-default-unsubscribed` (default 6), which are the levels that new groups will be put on if they are (un)subscribed. These two variables should, of course, be inside the relevant legal ranges.

If `gnus-keep-same-level` is non-`nil`, some movement commands will only move to groups that are of the same level (or lower). In particular, going from the last article in one group to the next group will go to the next group of the same level (or lower). This might be handy if you want to read the most important groups before you read the rest.

All groups with a level less than or equal to `gnus-group-default-list-level` will be listed in the group buffer by default.

If `gnus-group-list-inactive-groups` is non-`nil`, non-active groups will be listed along with the unread groups. This variable is `t` by default. If it is `nil`, inactive groups won't be listed.

If `gnus-group-use-permanent-levels` is non-`nil`, once you give a level prefix to *g* or *l*, all subsequent commands will use this level as the "work" level.

Gnus will normally just activate groups that are on level `gnus-activate-level` or less. If you don't want to activate unsubscribed groups, for instance, you might set this variable to `5`.

2.6 Group Score

You would normally keep important groups on high levels, but that scheme is somewhat restrictive. Don't you wish you could have Gnus sort the group buffer according to how often you read groups, perhaps? Within reason?

This is what *group score* is for. You can assign a score to each group. You can then sort the group buffer based on this score. Alternatively, you can sort on score and then level. (Taken together, the level and the score is called the *rank* of the group. A group that is on level 4 and has a score of 1 has a higher rank than a group on level 5 that has a score of 300. (The level is the most significant part and the score is the least significant part.)

If you want groups you read often to get higher scores than groups you read seldom you can add the `gnus-summary-bubble-group` function to the `gnus-summary-exit-hook` hook. This will result (after sorting) in a bubbling sort of action. If you want to see that in action after each summary exit, you can add `gnus-group-sort-groups-by-rank` or `gnus-group-sort-groups-by-score` to the same hook, but that will slow things down somewhat.

2.7 Marking Groups

If you want to perform some command on several groups, and they appear subsequently in the group buffer, you would normally just give a numerical prefix to the command. Most group commands will then do your bidding on those groups.

However, if the groups are not in sequential order, you can still perform a command on several groups. You simply mark the groups first with the process mark and then execute the command.

`#`
`M m` Set the mark on the current group (`gnus-group-mark-group`).

`M-#`
`M u` Remove the mark from the current group (`gnus-group-unmark-group`).

`M U` Remove the mark from all groups (`gnus-group-unmark-all-groups`).

`M w` Mark all groups between point and mark (`gnus-group-mark-region`).

`M b` Mark all groups in the buffer (`gnus-group-mark-buffer`).

`M r` Mark all groups that match some regular expression (`gnus-group-mark-regexp`).

Also see Section 8.1 [Process/Prefix], page 119.

If you want to execute some command on all groups that have been marked with the process mark, you can use the `M-&` (`gnus-group-universal-argument`) command. It will prompt you for the command to be executed.

2.8 Foreign Groups

Here are some group mode commands for making and editing general foreign groups, as well as commands to ease the creation of a few special-purpose groups:

G m Make a new group (`gnus-group-make-group`). Gnus will prompt you for a name, a method and possibly an *address*. For an easier way to subscribe to NNTP groups, see Section 2.13 [Browse Foreign Server], page 23.

G r Rename the current group to something else (`gnus-group-rename-group`). This is legal only on some groups – mail groups mostly. This command might very well be quite slow on some backends.

G e Enter a buffer where you can edit the select method of the current group (`gnus-group-edit-group-method`).

G p Enter a buffer where you can edit the group parameters (`gnus-group-edit-group-parameters`).

G E Enter a buffer where you can edit the group info (`gnus-group-edit-group`).

G d Make a directory group. You will be prompted for a directory name (`gnus-group-make-directory-group`).

G h Make the Gnus help group (`gnus-group-make-help-group`).

G a Make a Gnus archive group (`gnus-group-make-archive-group`). By default a group pointing to the most recent articles will be created (`gnus-group-recent-archive-directory`), but given a prefix, a full group will be created from from `gnus-group-archive-directory`.

G k Make a kiboze group. You will be prompted for a name, for a regexp to match groups to be "included" in the kiboze group, and a series of strings to match on headers (`gnus-group-make-kiboze-group`). See Section 6.5.2 [Kibozed Groups], page 100

G D Read an arbitrary directory as if with were a newsgroup with the `nneething` backend (`gnus-group-enter-directory`).

G f Make a group based on some file or other (`gnus-group-make-doc-group`). If you give a prefix to this command, you will be prompted for a file name and a file type. Currently supported types are `babyl`, `mbox`, `digest`, `mmdf`, `news`, `rnews`, `clari-briefs`, and `forward`. If you run this command without a prefix, Gnus will guess at the file type.

G DEL This function will delete the current group (`gnus-group-delete-group`). If given a prefix, this function will actually delete all the articles in the group, and forcibly remove the group itself from the face of the Earth. Use a prefix only if you are absolutely sure of what you are doing.

G V Make a new, fresh, empty `nnvirtual` group (`gnus-group-make-empty-virtual`).

G v Add the current group to an `nnvirtual` group (`gnus-group-add-to-virtual`). Uses the process/prefix convention.

See Chapter 6 [Select Methods], page 77 for more information on the various select methods.

If the `gnus-activate-foreign-newsgroups` is a positive number, Gnus will check all foreign groups with this level or lower at startup. This might take quite a while, especially if you subscribe to lots of groups from different NNTP servers.

2.9 Group Parameters

Gnus stores all information on a group in a list that is usually known as the *group info*. This list has from three to six elements. Here's an example info.

```
("nnml:mail.ding" 3 ((1 . 232) 244 (256 . 270)) ((tick 246 249))
                    (nnml "private") ((to-address . "ding@ifi.uio.no")))
```

The first element is the *group name*, as Gnus knows the group, anyway. The second element is the *subscription level*, which normally is a small integer. The third element is a list of ranges of read articles. The fourth element is a list of lists of article marks of various kinds. The fifth element is the select method (or virtual server, if you like). The sixth element is a list of *group parameters*, which is what this section is about.

Any of the last three elements may be missing if they are not required. In fact, the vast majority of groups will normally only have the first three elements, which saves quite a lot of cons cells.

The group parameters store information local to a particular group:

`to-address`
: If the group parameter list contains an element that looks like `(to-address . "some@where.com")`, that address will be used by the backend when doing followups and posts. This is primarily useful in mail groups that represent closed mailing lists—mailing lists where it's expected that everybody that writes to the mailing list is subscribed to it. Since using this parameter ensures that the mail only goes to the mailing list itself, it means that members won't receive two copies of your followups.

  Using `to-address` will actually work whether the group is foreign or not. Let's say there's a group on the server that is called '`fa.4ad-l`'. This is a real newsgroup, but the server has gotten the articles from a mail-to-news gateway. Posting directly to this group is therefore impossible—you have to send mail to the mailing list address instead.

`to-list`
: If the group parameter list has an element that looks like `(to-list . "some@where.com")`, that address will be used when doing a `a` in any group. It is totally ignored when doing a followup—except that if it is present in a news group, you'll get mail group semantics when doing `f`.

`broken-reply-to`
: Elements like `(broken-reply-to . t)` signals that `Reply-To` headers in this group are to be ignored. This can be useful if you're reading a mailing list group where the listserv has inserted `Reply-To` headers that point back to the listserv itself. This is broken behavior. So there!

`to-group` If the group parameter list contains an element like `(to-group . "some.group.name")`, all posts will be sent to that group.

`auto-expire`
If this symbol is present in the group parameter list, all articles that are read will be marked as expirable. For an alternative approach, see Section 6.3.7 [Expiring Mail], page 89.

`total-expire`
If this symbol is present, all read articles will be put through the expiry process, even if they are not marked as expirable. Use with caution.

`expiry-wait`
If the group parameter has an element that looks like `(expiry-wait . 10)`, this value will override any `nnmail-expiry-wait` and `nnmail-expiry-wait-function` when expiring expirable messages. The value can either be a number of days (not necessarily an integer) or the symbols `never` or `immediate`.

`score-file`
Elements that look like `(score-file . "file")` will make '`file`' into the current score file for the group in question. This means that all score commands you issue will end up in that file.

`admin-address`
When unsubscribing to a mailing list you should never send the unsubscription notice to the mailing list itself. Instead, you'd send messages to the administrative address. This parameter allows you to put the admin address somewhere convenient.

`comment` This parameter allows you to enter a arbitrary comment on the group.

(variable form)
You can use the group parameters to set variables local to the group you are entering. Say you want to turn threading off in '`news.answers`'. You'd then put `(gnus-show-threads nil)` in the group parameters of that group. `gnus-show-threads` will be made into a local variable in the summary buffer you enter, and the form `nil` will be `eval`ed there.

This can also be used as a group-specific hook function, if you'd like. If you want to hear a beep when you enter the group '`alt.binaries.pictures.furniture`', you could put something like `(dummy-variable (ding))` in the parameters of that group. `dummy-variable` will be set to the result of the `(ding)` form, but who cares?

If you want to change the group info you can use the *G E* command to enter a buffer where you can edit it.

You usually don't want to edit the entire group info, so you'd be better off using the *G p* command to just edit the group parameters.

2.10 Listing Groups

These commands all list various slices of the groups that are available.

`l`
`A s` List all groups that have unread articles (`gnus-group-list-groups`). If the numeric prefix is used, this command will list only groups of level ARG and lower. By default, it only lists groups of level five or lower (i.e., just subscribed groups).

`L`
`A u` List all groups, whether they have unread articles or not (`gnus-group-list-all-groups`). If the numeric prefix is used, this command will list only groups of level ARG and lower. By default, it lists groups of level seven or lower (i.e., just subscribed and unsubscribed groups).

`A l` List all unread groups on a specific level (`gnus-group-list-level`). If given a prefix, also list the groups with no unread articles.

`A k` List all killed groups (`gnus-group-list-killed`). If given a prefix argument, really list all groups that are available, but aren't currently (un)subscribed. This could entail reading the active file from the server.

`A z` List all zombie groups (`gnus-group-list-zombies`).

`A m` List all subscribed groups with unread articles that match a regexp (`gnus-group-list-matching`).

`A M` List groups that match a regexp (`gnus-group-list-all-matching`).

`A A` List absolutely all groups that are in the active file(s) of the server(s) you are connected to (`gnus-group-list-active`). This might very well take quite a while. It might actually be a better idea to do a `A m` to list all matching, and just give '`.`' as the thing to match on.

`A a` List all groups that have names that match a regexp (`gnus-group-apropos`).

`A d` List all groups that have names or descriptions that match a regexp (`gnus-group-description-apropos`).

Groups that match the `gnus-permanently-visible-groups` regexp will always be shown, whether they have unread articles or not. You can also add the `visible` element to the group parameters in question to get the same effect.

Groups that have just ticked articles in it are normally listed in the group buffer. If `gnus-list-groups-with-ticked-articles` is `nil`, these groups will be treated just like totally empty groups. It is `t` by default.

2.11 Sorting Groups

The `C-c C-s` (`gnus-group-sort-groups`) command sorts the group buffer according to the function(s) given by the `gnus-group-sort-function` variable. Available sorting functions include:

`gnus-group-sort-by-alphabet`
: Sort the group names alphabetically. This is the default.

`gnus-group-sort-by-level`
: Sort by group level.

`gnus-group-sort-by-score`
: Sort by group score.

`gnus-group-sort-by-rank`
: Sort by group score and then the group level. The level and the score are, when taken together, the group's *rank*.

`gnus-group-sort-by-unread`
: Sort by number of unread articles.

`gnus-group-sort-by-method`
: Sort by alphabetically on the select method.

`gnus-group-sort-function` can also be a list of sorting functions. In that case, the most significant sort key function must be the last one.

There are also a number of commands for sorting directly according to some sorting criteria:

G S a Sort the group buffer alphabetically by group name (`gnus-group-sort-groups-by-alphabet`).

G S u Sort the group buffer by the number of unread articles (`gnus-group-sort-groups-by-unread`).

G S l Sort the group buffer by group level (`gnus-group-sort-groups-by-level`).

G S v Sort the group buffer by group score (`gnus-group-sort-groups-by-score`).

G S r Sort the group buffer by group level (`gnus-group-sort-groups-by-rank`).

G S m Sort the group buffer alphabetically by backend name (`gnus-group-sort-groups-by-method`).

When given a prefix, all these commands will sort in reverse order.

2.12 Group Maintenance

b Find bogus groups and delete them (`gnus-group-check-bogus-groups`).

F Find new groups and process them (`gnus-find-new-newsgroups`). If given a prefix, use the `ask-server` method to query the server for new groups.

C-c C-x Run all expirable articles in the current group through the expiry process (if any) (`gnus-group-expire-articles`).

C-c M-C-x Run all articles in all groups through the expiry process (`gnus-group-expire-all-groups`).

2.13 Browse Foreign Server

B You will be queried for a select method and a server name. Gnus will then attempt to contact this server and let you browse the groups there (`gnus-group-browse-foreign-server`).

A new buffer with a list of available groups will appear. This buffer will be use the `gnus-browse-mode`. This buffer looks a bit (well, a lot) like a normal group buffer, but with one major difference - you can't enter any of the groups. If you want to read any of the news available on that server, you have to subscribe to the groups you think may be interesting, and then you have to exit this buffer. The new groups will be added to the group buffer, and then you can read them as you would any other group.

Future versions of Gnus may possibly permit reading groups straight from the browse buffer.

Here's a list of keystrokes available in the browse mode:

n Go to the next group (`gnus-group-next-group`).

p Go to the previous group (`gnus-group-prev-group`).

SPACE Enter the current group and display the first article (`gnus-browse-read-group`).

RET Enter the current group (`gnus-browse-select-group`).

u Unsubscribe to the current group, or, as will be the case here, subscribe to it (`gnus-browse-unsubscribe-current-group`).

l

q Exit browse mode (`gnus-browse-exit`).

? Describe browse mode briefly (well, there's not much to describe, is there) (`gnus-browse-describe-briefly`).

2.14 Exiting Gnus

Yes, Gnus is ex(c)iting.

z Suspend Gnus (`gnus-group-suspend`). This doesn't really exit Gnus, but it kills all buffers except the Group buffer. I'm not sure why this is a gain, but then who am I to judge?

q Quit Gnus (`gnus-group-exit`).

Q Quit Gnus without saving any startup files (`gnus-group-quit`).

`gnus-suspend-gnus-hook` is called when you suspend Gnus and `gnus-exit-gnus-hook` is called when you quit Gnus, while `gnus-after-exiting-gnus-hook` is called as the final item when exiting Gnus.

If you wish to completely unload Gnus and all its adherents, you can use the `gnus-unload` command. This command is also very handy when trying to customize meta-variables.

Note:

> Miss Lisa Cannifax, while sitting in English class, feels her feet go numbly heavy and herself fall into a hazy trance as the boy sitting behind her drew repeated lines with his pencil across the back of her plastic chair.

2.15 Group Topics

If you read lots and lots of groups, it might be convenient to group them hierarchically according to topics. You put your Emacs groups over here, your sex groups over there, and the rest (what, two groups or so?) you put in some misc section that you never bother with anyway. You can even group the Emacs sex groups as a sub-topic to either the Emacs groups or the sex groups—or both! Go wild!

To get this *fab* functionality you simply turn on (ooh!) the `gnus-topic` minor mode—type `t` in the group buffer. (This is a toggling command.)

Go ahead, just try it. I'll still be here when you get back. La de dum... Nice tune, that... la la la... What, you're back? Yes, and now press `l`. There. All your groups are now listed under '`misc`'. Doesn't that make you feel all warm and fuzzy? Hot and bothered?

If you want this permanently enabled, you should add that minor mode to the hook for the group mode:

```
(add-hook 'gnus-group-mode-hook 'gnus-topic-mode)
```

2.15.1 Topic Variables

Now, if you select a topic, if will fold/unfold that topic, which is really neat, I think.

The topic lines themselves are created according to the `gnus-topic-line-format` variable. See Section 8.3 [Formatting Variables], page 120. Elements allowed are:

'`i`' Indentation.

'`n`' Topic name.

'`v`' Visibility.

'`l`' Level.

'`g`' Number of groups in the topic.

'`a`' Number of unread articles in the topic.

'`A`' Number of unread articles in the topic and all its subtopics.

Each sub-topic (and the groups in the sub-topics) will be indented with `gnus-topic-indent-level` times the topic level number of spaces. The default is `2`.

`gnus-topic-mode-hook` is called in topic minor mode buffers.

2.15.2 Topic Commands

When the topic minor mode is turned on, a new *T* submap will be available. In addition, a few of the standard keys change their definitions slightly.

T n Prompt for a new topic name and create it (`gnus-topic-create-topic`).

T m Move the current group to some other topic (`gnus-topic-move-group`). This command understands the process/prefix convention (see Section 8.1 [Process/Prefix], page 119).

T c Copy the current group to some other topic (`gnus-topic-copy-group`). This command understands the process/prefix convention (see Section 8.1 [Process/Prefix], page 119).

T D Remove a group from the current topic (`gnus-topic-remove-group`). This command understands the process/prefix convention (see Section 8.1 [Process/Prefix], page 119).

T M Move all groups that match some regular expression to a topic (`gnus-topic-move-matching`).

T C Copy all groups that match some regular expression to a topic (`gnus-topic-copy-matching`).

T # Mark all groups in the current topic with the process mark (`gnus-topic-mark-topic`).

T M-# Remove the process mark from all groups in the current topic (`gnus-topic-unmark-topic`).

RET
SPACE Either select a group or fold a topic (`gnus-topic-select-group`). When you perform this command on a group, you'll enter the group, as usual. When done on a topic line, the topic will be folded (if it was visible) or unfolded (if it was folded already). So it's basically a toggling command on topics. In addition, if you give a numerical prefix, group on that level (and lower) will be displayed.

T TAB "Indent" the current topic so that it becomes a sub-topic of the previous topic (`gnus-topic-indent`). If given a prefix, "un-indent" the topic instead.

C-k Kill a group or topic (`gnus-topic-kill-group`).

C-y Yank the previously killed group or topic (`gnus-topic-yank-group`). Note that all topics will be yanked before all groups.

T r Rename a topic (`gnus-topic-rename`).

T DEL Delete an empty topic (`gnus-topic-delete`).

A T List all groups that Gnus knows about in a topics-ified way (`gnus-topic-list-active`).

2.15.3 Topic Topology

So, let's have a look at an example group buffer:

```
Gnus
  Emacs -- I wuw it!
       3: comp.emacs
       2: alt.religion.emacs
    Naughty Emacs
     452: alt.sex.emacs
       0: comp.talk.emacs.recovery
  Misc
       8: comp.binaries.fractals
      13: comp.sources.unix
```

So, here we have one top-level topic, two topics under that, and one sub-topic under one of the sub-topics. (There is always just one (1) top-level topic). This topology can be expressed as follows:

```
(("Gnus" visible)
 (("Emacs -- I wuw it!" visible)
  (("Naughty Emacs" visible)))
 (("Misc" visible)))
```

This is in fact how the variable `gnus-topic-topology` would look for the display above. That variable is saved in the '`.newsrc.eld`' file, and shouldn't be messed with manually—unless you really want to. Since this variable is read from the '`.newsrc.eld`' file, setting it in any other startup files will have no effect.

This topology shows what topics are sub-topics of what topics (right), and which topics are visible. Two settings are currently allowed—`visible` and `invisible`.

2.16 Misc Group Stuff

`^` Enter the server buffer (`gnus-group-enter-server-mode`). See Section 6.1 [The Server Buffer], page 77.

`a` Post an article to a group (`gnus-group-post-news`). The current group name will be used as the default.

m Mail a message somewhere (`gnus-group-mail`).

Variables for the group buffer:

`gnus-group-mode-hook`
: `gnus-group-mode-hook` is called after the group buffer has been created.

`gnus-group-prepare-hook`
: `gnus-group-prepare-hook` is called after the group buffer is generated. It may be used to modify the buffer in some strange, unnatural way.

`gnus-permanently-visible-groups`
: Groups matching this regexp will always be listed in the group buffer, whether they are empty or not.

2.16.1 Scanning New Messages

g : Check the server(s) for new articles. If the numerical prefix is used, this command will check only groups of level *arg* and lower (`gnus-group-get-new-news`). If given a non-numerical prefix, this command will force a total rereading of the active file(s) from the backend(s).

M-g : Check whether new articles have arrived in the current group (`gnus-group-get-new-news-this-group`). The `gnus-goto-next-group-when-activating` variable controls whether this command is to move point to the next group or not. It is `t` by default.

C-c M-g : Activate absolutely all groups (`gnus-activate-all-groups`).

R : Restart Gnus (`gnus-group-restart`).

`gnus-get-new-news-hook` is run just before checking for new news.

`gnus-after-getting-new-news-hook` is run after checking for new news.

2.16.2 Group Information

M-f : Try to fetch the FAQ for the current group (`gnus-group-fetch-faq`). Gnus will try to get the FAQ from `gnus-group-faq-directory`, which is usually a directory on a remote machine. `ange-ftp` will be used for fetching the file.

D : Describe the current group (`gnus-group-describe-group`). If given a prefix, force Gnus to re-read the description from the server.

M-d : Describe all groups (`gnus-group-describe-all-groups`). If given a prefix, force Gnus to re-read the description file from the server.

V : Display current Gnus version numbers (`gnus-version`).

? : Give a very short help message (`gnus-group-describe-briefly`).

C-c C-i : Go to the Gnus info node (`gnus-info-find-node`).

2.16.3 File Commands

r : Read the init file (`gnus-init-file`, which defaults to '`~/.gnus`') (`gnus-group-read-init-file`).

s : Save the '`.newsrc.eld`' file (and '`.newsrc`' if wanted) (`gnus-group-save-newsrc`). If given a prefix, force saving the file(s) whether Gnus thinks it is necessary or not.

3 The Summary Buffer

A line for each article is displayed in the summary buffer. You can move around, read articles, post articles and reply to articles.

3.1 Summary Buffer Format

Gnus will use the value of the `gnus-extract-address-components` variable as a function for getting the name and address parts of a `From` header. Two pre-defined function exist: `gnus-extract-address-components`, which is the default, quite fast, and too simplistic solution; and `mail-extract-address-components`, which works very nicely, but is slower. The default function will return the wrong answer in 5% of the cases. If this is unacceptable to you, use the other function instead.

`gnus-summary-same-subject` is a string indicating that the current article has the same subject as the previous. This string will be used with those specs that require it. The default is ‘’.

3.1.1 Summary Buffer Lines

You can change the format of the lines in the summary buffer by changing the `gnus-summary-line-format` variable. It works along the same lines a a normal `format` string, with some extensions.

The default string is ‘`%U%R%z%I%(%[%4L: %-20,20n%]%) %s\n`’.

The following format specification characters are understood:

‘`N`’ Article number.

‘`S`’ Subject string.

‘`s`’ Subject if the article is the root, `gnus-summary-same-subject` otherwise.

‘`F`’ Full `From` line.

‘`n`’ The name (from the `From` header).

‘`a`’ The name (from the `From` header). This differs from the `n` spec in that it uses `gnus-extract-address-components`, which is slower, but may be more thorough.

‘`A`’ The address (from the `From` header). This works the same way as the `a` spec.

‘`L`’ Number of lines in the article.

‘`c`’ Number of characters in the article.

‘`I`’ Indentation based on thread level (see Section 3.9.1 [Customizing Threading], page 41).

‘T’ Nothing if the article is a root and lots of spaces if it isn't (it pushes everything after it off the screen).

‘\[’ Opening bracket, which is normally ‘\[’, but can also be ‘<’ for adopted articles.

‘\]’ Closing bracket, which is normally ‘\]’, but can also be ‘>’ for adopted articles.

‘>’ One space for each thread level.

‘<’ Twenty minus thread level spaces.

‘U’ Unread.

‘R’ Replied.

‘i’ Score as a number.

‘z’ Zcore, ‘+’ if above the default level and ‘-’ if below the default level. If the difference between `gnus-summary-default-level` and the score is less than `gnus-summary-zcore-fuzz`, this spec will not be used.

‘V’ Total thread score.

‘x’ `Xref`.

‘D’ `Date`.

‘M’ `Message-ID`.

‘r’ `References`.

‘t’ Number of articles in the current sub-thread. Using this spec will slow down summary buffer generation somewhat.

‘e’ A single character will be displayed if the article has any children.

‘u’ User defined specifier. The next character in the format string should be a letter. GNUS will call the function `gnus-user-format-function-`‘X’, where ‘X’ is the letter following ‘%u’. The function will be passed the current header as argument. The function should return a string, which will be inserted into the summary just like information from any other summary specifier.

The ‘%U’ (status), ‘%R’ (replied) and ‘%z’ (zcore) specs have to be handled with care. For reasons of efficiency, Gnus will compute what column these characters will end up in, and “hard-code” that. This means that it is illegal to have these specs after a variable-length spec. Well, you might not be arrested, but your summary buffer will look strange, which is bad enough.

The smart choice is to have these specs as far to the left as possible. (Isn't that the case with everything, though? But I digress.)

This restriction may disappear in later versions of Gnus.

3.1.2 Summary Buffer Mode Line

You can also change the format of the summary mode bar. Set `gnus-summary-mode-line-format` to whatever you like. Here are the elements you can play with:

‘G’ Group name.

‘p’ Unprefixed group name.

‘A’ Current article number.

‘V’ Gnus version.

‘U’ Number of unread articles in this group.

‘e’ Number of unselected articles in this group.

‘Z’ A string with the number of unread and unselected articles represented either as ‘<%U(+%u) more>’ if there are both unread and unselected articles, and just as ‘<%U more>’ if there are just unread articles and no unselected ones.

‘g’ Shortish group name. For instance, ‘rec.arts.anime’ will be shortened to ‘r.a.anime’.

‘S’ Subject of the current article.

‘u’ Used-defined spec.

‘s’ Name of the current score file.

‘d’ Number of dormant articles.

‘t’ Number of ticked articles.

‘r’ Number of articles that have been marked as read in this session.

‘E’ Number of articles expunged by the score files.

3.1.3 Summary Highlighting

`gnus-visual-mark-article-hook`
: This hook is run after selecting an article. It is meant to be used for highlighting the article in some way. It is not run if `gnus-visual` is `nil`.

`gnus-summary-update-hook`
: This hook is called when a summary line is changed. It is not run if `gnus-visual` is `nil`.

`gnus-summary-selected-face`
: This is the face (or *font* as some people call it) that is used to highlight the current article in the summary buffer.

`gnus-summary-highlight`
: Summary lines are highlighted according to this variable, which is a list where the elements are on the format `(FORM . FACE)`. If you would, for instance, like ticked articles to be italic and high-scored articles to be bold, you could set this variable to something like

```
(((eq mark gnus-ticked-mark) . italic)
 ((> score default) . bold))
```

: As you may have guessed, if *FORM* returns a non-`nil` value, *FACE* will be applied to the line.

3.2 Summary Maneuvering

All the straight movement commands understand the numeric prefix and behave pretty much as you'd expect.

None of these commands select articles.

G M-n
M-n Go to the next summary line of an unread article (`gnus-summary-next-unread-subject`).

G M-p
M-p Go to the previous summary line of an unread article (`gnus-summary-prev-unread-subject`).

G j
j Ask for an article number and then go that article (`gnus-summary-goto-article`).

G g Ask for an article number and then go the summary line of that article (`gnus-summary-goto-subject`).

If Gnus asks you to press a key to confirm going to the next group, you can use the *C-n* and *C-p* keys to move around the group buffer, searching for the next group to read without actually returning to the group buffer.

Variables related to summary movement:

`gnus-auto-select-next`
If you are at the end of the group and issue one of the movement commands, Gnus will offer to go to the next group. If this variable is `t` and the next group is empty, Gnus will exit summary mode and return to the group buffer. If this variable is neither `t` nor `nil`, Gnus will select the next group, no matter whether it has any unread articles or not. As a special case, if this variable is `quietly`, Gnus will select the next group without asking for confirmation. If this variable is `almost-quietly`, the same will happen only if you are located on the last article in the group. Finally, if this variable is `slightly-quietly`, the *Z n* command will go to the next group without confirmation. Also see Section 2.5 [Group Levels], page 16.

`gnus-auto-select-same`
If non-`nil`, all the movement commands will try to go to the next article with the same subject as the current. This variable is not particularly useful if you use a threaded display.

`gnus-summary-check-current`
If non-`nil`, all the "unread" movement commands will not proceed to the next (or previous) article if the current article is unread. Instead, they will choose the current article.

`gnus-auto-center-summary`
If non-`nil`, Gnus will keep the point in the summary buffer centered at all times. This makes things quite tidy, but if you have a slow network connection,

or simply do not like this un-Emacsism, you can set this variable to `nil` to get the normal Emacs scrolling action. This will also inhibit horizontal re-centering of the summary buffer, which might make it more inconvenient to read extremely long threads.

3.3 Choosing Articles

None of the following movement commands understand the numeric prefix, and they all select and display an article.

SPACE Select the current article, or, if that one's read already, the next unread article (`gnus-summary-next-page`).

G n
n Go to next unread article (`gnus-summary-next-unread-article`).

G p
p Go to previous unread article (`gnus-summary-prev-unread-article`).

G N
N Go to the next article (`gnus-summary-next-article`).

G P
P Go to the previous article (`gnus-summary-prev-article`).

G C-n Go to the next article with the same subject (`gnus-summary-next-same-subject`).

G C-p Go to the previous article with the same subject (`gnus-summary-prev-same-subject`).

G f
. Go to the first unread article (`gnus-summary-first-unread-article`).

G b
, Go to the article with the highest score (`gnus-summary-best-unread-article`).

G l
l Go to the previous article read (`gnus-summary-goto-last-article`).

G p Pop an article off the summary history and go to this article (`gnus-summary-pop-article`). This command differs from the command above in that you can pop as many previous articles off the history as you like.

Some variables that are relevant for moving and selecting articles:

`gnus-auto-extend-newsgroup`
All the movement commands will try to go to the previous (or next) article, even if that article isn't displayed in the Summary buffer if this variable is non-`nil`. Gnus will then fetch the article from the server and display it in the article buffer.

`gnus-select-article-hook`
: This hook is called whenever an article is selected. By default it exposes any threads hidden under the selected article.

`gnus-mark-article-hook`
: This hook is called whenever an article is selected. It is intended to be used for marking articles as read. The default value is `gnus-summary-mark-read-and-unread-as-read`, and will change the mark of almost any article you read to `gnus-unread-mark`. The only articles not affected by this function are ticked, dormant, and expirable articles. If you'd instead like to just have unread articles marked as read, you can use `gnus-summary-mark-unread-as-read` instead. It will leave marks like `gnus-low-score-mark`, `gnus-del-mark` (and so on) alone.

3.4 Scrolling the Article

SPACE
: Pressing *SPACE* will scroll the current article forward one page, or, if you have come to the end of the current article, will choose the next article (`gnus-summary-next-page`).

DEL
: Scroll the current article back one page (`gnus-summary-prev-page`).

RET
: Scroll the current article one line forward (`gnus-summary-scroll-up`).

A g
g
: (Re)fetch the current article (`gnus-summary-show-article`). If given a prefix, fetch the current article, but don't run any of the article treatment functions. This will give you a "raw" article, just the way it came from the server.

A <
<
: Scroll to the beginning of the article (`gnus-summary-beginning-of-article`).

A >
>
: Scroll to the end of the article (`gnus-summary-end-of-article`).

A s
: Perform an isearch in the article buffer (`gnus-summary-isearch-article`).

3.5 Reply, Followup and Post

3.5.1 Summary Mail Commands

Commands for composing a mail message:

S r
r
: Mail a reply to the author of the current article (`gnus-summary-reply`).

S R
R
: Mail a reply to the author of the current article and include the original message (`gnus-summary-reply-with-original`). This command uses the process/prefix convention.

`S o m` Forward the current article to some other person (`gnus-summary-mail-forward`).

`S o p` Forward the current article to a newsgroup (`gnus-summary-post-forward`).

`S m`
`m` Send a mail to some other person (`gnus-summary-mail-other-window`).

`S D b` If you have sent a mail, but the mail was bounced back to you for some reason (wrong address, transient failure), you can use this command to resend that bounced mail (`gnus-summary-resend-bounced-mail`). You will be popped into a mail buffer where you can edit the headers before sending the mail off again. If you give a prefix to this command, and the bounced mail is a reply to some other mail, Gnus will try to fetch that mail and display it for easy perusal of its headers. This might very well fail, though.

`S D r` Not to be confused with the previous command, `gnus-summary-resend-message` will prompt you for an address to send the current message off to, and then send it to that place. The headers of the message won't be altered—but lots of headers that say `Resent-To`, `Resent-From` and so on will be added. This means that you actually send a mail to someone that has a `To` header that (probably) points to yourself. This will confuse people. So, natcherly you'll only do that if you're really eVIl.

This command is mainly used if you have several accounts and want to ship a mail to a different account of yours. (If you're both `root` and `postmaster` and get a mail for `postmaster` to the `root` account, you may want to resend it to `postmaster`. Ordnung muss sein!

`S O m` Digest the current series and forward the result using mail (`gnus-uu-digest-mail-forward`). This command uses the process/prefix convention (see Section 8.1 [Process/Prefix], page 119).

`S O p` Digest the current series and forward the result to a newsgroup (`gnus-uu-digest-mail-forward`).

3.5.2 Summary Post Commands

Commands for posting an article:

`S p`
`a` Post an article to the current group (`gnus-summary-post-news`).

`S f`
`f` Post a followup to the current article (`gnus-summary-followup`).

`S F`
`F` Post a followup to the current article and include the original message (`gnus-summary-followup-with-original`). This command uses the process/prefix convention.

`S u` Uuencode a file, split it into parts, and post it as a series (`gnus-uu-post-news`). (see Section 3.16.4.3 [Uuencoding and Posting], page 54).

3.6 Canceling Articles

Have you ever written something, and then decided that you really, really, really wish you hadn't posted that?

Well, you can't cancel mail, but you can cancel posts.

Find the article you wish to cancel (you can only cancel your own articles, so don't try any funny stuff). Then press `C` or `S c` (`gnus-summary-cancel-article`). Your article will be canceled—machines all over the world will be deleting your article.

Be aware, however, that not all sites honor cancels, so your article may live on here and there, while most sites will delete the article in question.

If you discover that you have made some mistakes and want to do some corrections, you can post a *superseding* article that will replace your original article.

Go to the original article and press `S s` (`gnus-summary-supersede-article`). You will be put in a buffer where you can edit the article all you want before sending it off the usual way.

The same goes for superseding as for canceling, only more so: Some sites do not honor superseding. On those sites, it will appear that you have posted almost the same article twice.

If you have just posted the article, and change your mind right away, there is a trick you can use to cancel/supersede the article without waiting for the article to appear on your site first. You simply return to the post buffer (which is called `*post-buf*`). There you will find the article you just posted, with all the headers intact. Change the `Message-ID` header to a `Cancel` or `Supersedes` header by substituting one of those words for `Message-ID`. Then just press `C-c C-c` to send the article as you would do normally. The previous article will be canceled/superseded.

Just remember, kids: There is no 'c' in 'supersede'.

3.7 Marking Articles

There are several marks you can set on an article.

You have marks that decide the *readedness* (whoo, neato-keano neologism ohoy!) of the article. Alphabetic marks generally mean *read*, while non-alphabetic characters generally mean *unread*.

In addition, you also have marks that do not affect readedness.

3.7.1 Unread Articles

The following marks mark articles as unread, in one form or other.

'!' *Ticked articles* are articles that will remain visible always. If you see an article that you find interesting, or you want to put off reading it, or replying to it, until sometime later, you'd typically tick it. However, articles can be expired,

so if you want to keep an article forever, you'll have to save it. Ticked articles have a '`!`' (`gnus-ticked-mark`) in the first column.

'`?`' A *dormant* article is marked with a '`?`' (`gnus-dormant-mark`), and will only appear in the summary buffer if there are followups to it.

'`SPACE`' An *unread* article is marked with a '`SPACE`' (`gnus-unread-mark`). These are articles that haven't been read at all yet.

3.7.2 Read Articles

All the following marks mark articles as read.

'`r`' Articles that are marked as read. They have a '`r`' (`gnus-del-mark`) in the first column. These are articles that the user has marked as read more or less manually.

'`R`' Articles that are actually read are marked with '`R`' (`gnus-read-mark`).

'`O`' Articles that were marked as read in previous sessions are now *old* and marked with '`O`' (`gnus-ancient-mark`).

'`K`' Marked as killed (`gnus-killed-mark`).

'`X`' Marked as killed by kill files (`gnus-kill-file-mark`).

'`Y`' Marked as read by having a too low score (`gnus-low-score-mark`).

'`C`' Marked as read by a catchup (`gnus-catchup-mark`).

'`G`' Canceled article (`gnus-canceled-mark`)

'`F`' SOUPed article (`gnus-souped-mark`).

'`Q`' Sparsely reffed article (`gnus-sparse-mark`).

All these marks just mean that the article is marked as read, really. They are interpreted differently by the adaptive scoring scheme, however.

One more special mark, though:

'`E`' You can also mark articles as *expirable* (or have them marked as such automatically). That doesn't make much sense in normal groups, because a user does not control the expiring of news articles, but in mail groups, for instance, articles that are marked as *expirable* can be deleted by Gnus at any time. Expirable articles are marked with '`E`' (`gnus-expirable-mark`).

3.7.3 Other Marks

There are some marks that have nothing to do with whether the article is read or not.

- You can set a bookmark in the current article. Say you are reading a long thesis on cats' urinary tracts, and have to go home for dinner before you've finished reading the thesis. You can then set a bookmark in the article, and Gnus will jump to this bookmark the next time it encounters the article.

- All articles that you have replied to or made a followup to (i.e., have answered) will be marked with an '`A`' in the second column (`gnus-replied-mark`).
- Articles that are stored in the article cache will be marked with an '`*`' in the second column (`gnus-cached-mark`).
- Articles that are "saved" (in some manner or other; not necessarily religiously) are marked with an '`S`' in the second column (`gnus-saved-mark`.
- It the '`%e`' spec is used, the presence of threads or not will be marked with `gnus-not-empty-thread-mark` and `gnus-empty-thread-mark` in the third column, respectively.
- Finally we have the *process mark* (`gnus-process-mark`. A variety of commands react to the presence of the process mark. For instance, *X u* (`gnus-uu-decode-uu`) will uudecode and view all articles that have been marked with the process mark. Articles marked with the process mark have a '`#`' in the second column.

You might have noticed that most of these "non-readedness" marks appear in the second column by default. So if you have a cached, saved, replied article that you have process-marked, what will that look like?

Nothing much. The precedence rules go as follows: process -> cache -> replied -> saved. So if the article is in the cache and is replied, you'll only see the cache mark and not the replied mark.

3.7.4 Setting Marks

All the marking commands understand the numeric prefix.

M t
! Tick the current article (`gnus-summary-tick-article-forward`).

M ?
? Mark the current article as dormant (`gnus-summary-mark-as-dormant`).

M d
d Mark the current article as read (`gnus-summary-mark-as-read-forward`).

M k
k Mark all articles that have the same subject as the current one as read, and then select the next unread article (`gnus-summary-kill-same-subject-and-select`).

M K
C-k Mark all articles that have the same subject as the current one as read (`gnus-summary-kill-same-subject`).

M C Mark all unread articles in the group as read (`gnus-summary-catchup`).

M C-c Mark all articles in the group as read—even the ticked and dormant articles (`gnus-summary-catchup-all`).

M H Catchup the current group to point (`gnus-summary-catchup-to-here`).

C-w Mark all articles between point and mark as read (`gnus-summary-mark-region-as-read`).

M V k Kill all articles with scores below the default score (or below the numeric prefix) (`gnus-summary-kill-below`).

M c
M-u Clear all readedness-marks from the current article (`gnus-summary-clear-mark-forward`).

M e
E Mark the current article as expirable (`gnus-summary-mark-as-expirable`).

M b Set a bookmark in the current article (`gnus-summary-set-bookmark`).

M B Remove the bookmark from the current article (`gnus-summary-remove-bookmark`).

M V c Clear all marks from articles with scores over the default score (or over the numeric prefix) (`gnus-summary-clear-above`).

M V u Tick all articles with scores over the default score (or over the numeric prefix) (`gnus-summary-tick-above`).

M V m Prompt for a mark, and mark all articles with scores over the default score (or over the numeric prefix) with this mark (`gnus-summary-clear-above`).

The `gnus-summary-goto-unread` variable controls what action should be taken after setting a mark. If non-`nil`, point will move to the next/previous unread article. If `nil`, point will just move one line up or down. As a special case, if this variable is `never`, all the marking commands as well as other commands (like *SPACE*) will move to the next article, whether it is unread or not. The default is `t`.

3.7.5 Setting Process Marks

M P p
Mark the current article with the process mark (`gnus-summary-mark-as-processable`).

M P u
M-# Remove the process mark, if any, from the current article (`gnus-summary-unmark-as-processable`).

M P U Remove the process mark from all articles (`gnus-summary-unmark-all-processable`).

M P R Mark articles by a regular expression (`gnus-uu-mark-by-regexp`).

M P r Mark articles in region (`gnus-uu-mark-region`).

M P t Mark all articles in the current (sub)thread (`gnus-uu-mark-thread`).

M P T Unmark all articles in the current (sub)thread (`gnus-uu-unmark-thread`).

M P v Mark all articles that have a score above the prefix argument (`gnus-uu-mark-over`).

M P s Mark all articles in the current series (`gnus-uu-mark-series`).

`M P S` Mark all series that have already had some articles marked (`gnus-uu-mark-sparse`).

`M P a` Mark all articles in series order (`gnus-uu-mark-series`).

`M P b` Mark all articles in the buffer in the order they appear (`gnus-uu-mark-buffer`).

3.8 Limiting

It can be convenient to limit the summary buffer to just show some subset of the articles currently in the group. The effect most limit commands have is to remove a few (or many) articles from the summary buffer.

`/ /`
`/ s` Limit the summary buffer to articles that match some subject (`gnus-summary-limit-to-subject`).

`/ a` Limit the summary buffer to articles that match some author (`gnus-summary-limit-to-author`).

`/ u`
`x` Limit the summary buffer to articles that are not marked as read (`gnus-summary-limit-to-unread`). If given a prefix, limit the buffer to articles that are strictly unread. This means that ticked and dormant articles will also be excluded.

`/ m` Ask for a mark and then limit to all articles that have not been marked with that mark (`gnus-summary-limit-to-marks`).

`/ n` Limit the summary buffer to the current article (`gnus-summary-limit-to-articles`). Uses the process/prefix convention (see Section 8.1 [Process/Prefix], page 119).

`/ w` Pop the previous limit off the stack and restore it (`gnus-summary-pop-limit`). If given a prefix, pop all limits off the stack.

`/ v` Limit the summary buffer to articles that have a score at or above some score (`gnus-summary-limit-to-score`).

`/ E`
`M S` Display all expunged articles (`gnus-summary-limit-include-expunged`).

`/ D` Display all dormant articles (`gnus-summary-limit-include-dormant`).

`/ d` Hide all dormant articles (`gnus-summary-limit-exclude-dormant`).

`/ c` Hide all dormant articles that have no children (`gnus-summary-limit-exclude-childless-dormant`).

`/ C` Mark all excluded unread articles as read (`gnus-summary-limit-mark-excluded-as-read`). If given a prefix, also mark excluded ticked and dormant articles as read.

3.9 Threading

Gnus threads articles by default. *To thread* is to put replies to articles directly after the articles they reply to—in a hierarchical fashion.

3.9.1 Customizing Threading

`gnus-show-threads`
: If this variable is `nil`, no threading will be done, and all of the rest of the variables here will have no effect. Turning threading off will speed group selection up a bit, but it is sure to make reading slower and more awkward.

`gnus-fetch-old-headers`
: If non-`nil`, Gnus will attempt to build old threads by fetching more old headers—headers to articles that are marked as read. If you would like to display as few summary lines as possible, but still connect as many loose threads as possible, you should set this variable to `some` or a number. If you set it to a number, no more than that number of extra old headers will be fetched. In either case, fetching old headers only works if the backend you are using carries overview files—this would normally be `nntp`, `nnspool` and `nnml`. Also remember that if the root of the thread has been expired by the server, there's not much Gnus can do about that.

`gnus-build-sparse-threads`
: Fetching old headers can be slow. A low-rent similar effect can be gotten by setting this variable to `some`. Gnus will then look at the complete `References` headers of all articles and try to string articles that belong in the same thread together. This will leave *gaps* in the threading display where Gnus guesses that an article is missing from the thread. (These gaps appear like normal summary lines. If you select a gap, Gnus will try to fetch the article in question.) If this variable is `t`, Gnus will display all these "gaps" without regard for whether they are useful for completing the thread or not. Finally, if this variable is `more`, Gnus won't cut off sparse leaf nodes that don't lead anywhere. This variable is `nil` by default.

`gnus-summary-gather-subject-limit`
: Loose threads are gathered by comparing subjects of articles. If this variable is `nil`, Gnus requires an exact match between the subjects of the loose threads before gathering them into one big super-thread. This might be too strict a requirement, what with the presence of stupid newsreaders that chop off long subjects lines. If you think so, set this variable to, say, 20 to require that only the first 20 characters of the subjects have to match. If you set this variable to a really low number, you'll find that Gnus will gather everything in sight into one thread, which isn't very helpful.

  If you set this variable to the special value `fuzzy`, Gnus will use a fuzzy string comparison algorithm on the subjects.

`gnus-simplify-subject-fuzzy-regexp`
: This can either be a regular expression or list of regular expressions that match strings that will be removed from subjects if fuzzy subject simplification is used.

`gnus-simplify-ignored-prefixes`
: If you set `gnus-summary-gather-subject-limit` to something as low as 10, you might consider setting this variable to something sensible:

```
(setq gnus-simplify-ignored-prefixes
      (concat
       "\\`\\[?\\("
       (mapconcat 'identity
                  '("looking"
                    "wanted" "followup" "summary\\( of\\)?"
                    "help" "query" "problem" "question"
                    "answer" "reference" "announce"
                    "How can I" "How to" "Comparison of"
                    ;; ...
                    )
                  "\\|")
       "\\)\\s *\\("
       (mapconcat 'identity
                  '("for" "for reference" "with" "about")
                  "\\|")
       "\\)?\\]?:?[ \t]*"))
```

: All words that match this regexp will be removed before comparing two subjects.

`gnus-summary-gather-exclude-subject`
: Since loose thread gathering is done on subjects only, that might lead to many false hits, especially with certain common subjects like ‘’ and ‘`(none)`’. To make the situation slightly better, you can use the regexp `gnus-summary-gather-exclude-subject` to say what subjects should be excluded from the gathering process. The default is ‘`^ *$\\|^(none)$`’.

`gnus-summary-thread-gathering-function`
: Gnus gathers threads by looking at `Subject` headers. This means that totally unrelated articles may end up in the same “thread”, which is confusing. An alternate approach is to look at all the `Message-ID`s in all the `References` headers to find matches. This will ensure that no gathered threads ever includes unrelated articles, but it's also means that people who have posted with broken newsreaders won't be gathered properly. The choice is yours—plague or cholera:

  `gnus-gather-threads-by-subject`
  : This function is the default gathering function and looks at `Subject`s exclusively.

  `gnus-gather-threads-by-references`
  : This function looks at `References` headers exclusively.

: If you want to test gathering by `References`, you could say something like:

```
(setq gnus-summary-thread-gathering-function
      'gnus-gather-threads-by-references)
```

`gnus-summary-make-false-root`
: If non-`nil`, Gnus will gather all loose subtrees into one big tree and create a dummy root at the top. (Wait a minute. Root at the top? Yup.) Loose subtrees occur when the real root has expired, or you've read or killed the root in a previous session.

  When there is no real root of a thread, Gnus will have to fudge something. This variable says what fudging method Gnus should use. There are four possible values:

  `adopt`
  : Gnus will make the first of the orphaned articles the parent. This parent will adopt all the other articles. The adopted articles will be marked as such by pointy brackets (`<>`) instead of the standard square brackets (`[]`). This is the default method.

  `dummy`
  : Gnus will create a dummy summary line that will pretend to be the parent. This dummy line does not correspond to any real article, so selecting it will just select the first real article after the dummy article. `gnus-summary-dummy-line-format` is used to specify the format of the dummy roots. It accepts only one format spec: `S`, which is the subject of the article. See Section 8.3 [Formatting Variables], page 120.

  `empty`
  : Gnus won't actually make any article the parent, but simply leave the subject field of all orphans except the first empty. (Actually, it will use `gnus-summary-same-subject` as the subject (see Section 3.1 [Summary Buffer Format], page 29).)

  `none`
  : Don't make any article parent at all. Just gather the threads and display them after one another.

  `nil`
  : Don't gather loose threads.

`gnus-thread-hide-subtree`
: If non-`nil`, all threads will be hidden when the summary buffer is generated.

`gnus-thread-hide-killed`
: if you kill a thread and this variable is non-`nil`, the subtree will be hidden.

`gnus-thread-ignore-subject`
: Sometimes somebody changes the subject in the middle of a thread. If this variable is non-`nil`, the subject change is ignored. If it is `nil`, which is the default, a change in the subject will result in a new thread.

`gnus-thread-indent-level`
: This is a number that says how much each sub-thread should be indented. The default is `4`.

3.9.2 Thread Commands

T k
M-C-k Mark all articles in the current sub-thread as read (`gnus-summary-kill-thread`). If the prefix argument is positive, remove all marks instead. If the prefix argument is negative, tick articles instead.

T l
M-C-l Lower the score of the current thread (`gnus-summary-lower-thread`).

T i Increase the score of the current thread (`gnus-summary-raise-thread`).

T # Set the process mark on the current thread (`gnus-uu-mark-thread`).

T M-# Remove the process mark from the current thread (`gnus-uu-unmark-thread`).

T T Toggle threading (`gnus-summary-toggle-threads`).

T s Expose the thread hidden under the current article, if any (`gnus-summary-show-thread`).

T h Hide the current (sub)thread (`gnus-summary-hide-thread`).

T S Expose all hidden threads (`gnus-summary-show-all-threads`).

T H Hide all threads (`gnus-summary-hide-all-threads`).

T t Re-thread the thread the current article is part of (`gnus-summary-rethread-current`). This works even when the summary buffer is otherwise unthreaded.

T ^ Make the current article the child of the marked (or previous) article (`gnus-summary-reparent-thread`.

The following commands are thread movement commands. They all understand the numeric prefix.

T n Go to the next thread (`gnus-summary-next-thread`).

T p Go to the previous thread (`gnus-summary-prev-thread`).

T d Descend the thread (`gnus-summary-down-thread`).

T u Ascend the thread (`gnus-summary-up-thread`).

T o Go to the top of the thread (`gnus-summary-top-thread`).

If you ignore subject while threading, you'll naturally end up with threads that have several different subjects in them. If you then issue a command like 'T k' (`gnus-summary-kill-thread`) you might not wish to kill the entire thread, but just those parts of the thread that have the same subject as the current article. If you like this idea, you can fiddle with `gnus-thread-operation-ignore-subject`. If is is non-`nil` (which it is by default), subjects will be ignored when doing thread commands. If this variable is `nil`, articles in the same thread with different subjects will not be included in the operation in question. If this variable is `fuzzy`, only articles that have subjects that are fuzzily equal will be included.

3.10 Sorting

If you are using a threaded summary display, you can sort the threads by setting `gnus-thread-sort-functions`, which is a list of functions. By default, sorting is done on article numbers. Ready-made sorting predicate functions include `gnus-thread-sort-by-number`, `gnus-thread-sort-by-author`, `gnus-thread-sort-by-subject`, `gnus-thread-sort-by-date`, `gnus-thread-sort-by-score`, and `gnus-thread-sort-by-total-score`.

Each function takes two threads and return non-`nil` if the first thread should be sorted before the other. Note that sorting really is normally done by looking only at the roots of each thread. If you use more than one function, the primary sort key should be the last function in the list. You should probably always include `gnus-thread-sort-by-number` in the list of sorting functions—preferably first. This will ensure that threads that are equal with respect to the other sort criteria will be displayed in ascending article order.

If you would like to sort by score, then by subject, and finally by number, you could do something like:

```
(setq gnus-thread-sort-functions
      '(gnus-thread-sort-by-number
        gnus-thread-sort-by-subject
        gnus-thread-sort-by-score))
```

The threads that have highest score will be displayed first in the summary buffer. When threads have the same score, they will be sorted alphabetically. The threads that have the same score and the same subject will be sorted by number, which is (normally) the sequence in which the articles arrived.

If you want to sort by score and then reverse arrival order, you could say something like:

```
(setq gnus-thread-sort-functions
      '((lambda (t1 t2)
          (not (gnus-thread-sort-by-number t1 t2)))
        gnus-thread-sort-by-score))
```

The function in the `gnus-thread-score-function` variable (default `+`) is used for calculating the total score of a thread. Useful functions might be `max`, `min`, or squared means, or whatever tickles your fancy.

If you are using an unthreaded display for some strange reason or other, you have to fiddle with the `gnus-article-sort-functions` variable. It is very similar to the `gnus-thread-sort-functions`, except that is uses slightly different functions for article comparison. Available sorting predicate functions are `gnus-article-sort-by-number`, `gnus-article-sort-by-author`, `gnus-article-sort-by-subject`, `gnus-article-sort-by-date`, and `gnus-article-sort-by-score`.

If you want to sort an unthreaded summary display by subject, you could say something like:

```
(setq gnus-article-sort-functions
      '(gnus-article-sort-by-number
        gnus-article-sort-by-subject))
```

3.11 Asynchronous Article Fetching

If you read your news from an NNTP server that's far away, the network latencies may make reading articles a chore. You have to wait for a while after pressing `n` to go to the next article before the article appears. Why can't Gnus just go ahead and fetch the article while you are reading the previous one? Why not, indeed.

First, some caveats. There are some pitfalls to using asynchronous article fetching, especially the way Gnus does it.

Let's say you are reading article 1, which is short, and article 2 is quite long, and you are not interested in reading that. Gnus does not know this, so it goes ahead and fetches article 2. You decide to read article 3, but since Gnus is in the process of fetching article 2, the connection is blocked.

To avoid these situations, Gnus will open two (count 'em two) connections to the server. Some people may think this isn't a very nice thing to do, but I don't see any real alternatives. Setting up that extra connection takes some time, so Gnus startup will be slower.

Gnus will fetch more articles than you will read. This will mean that the link between your machine and the NNTP server will become more loaded than if you didn't use article pre-fetch. The server itself will also become more loaded—both with the extra article requests, and the extra connection.

Ok, so now you know that you shouldn't really use this thing... unless you really want to.

Here's how: Set `gnus-asynchronous` to `t`. The rest should happen automatically.

You can control how many articles that are to be pre-fetched by setting `nntp-async-number`. This is five by default, which means that when you read an article in the group, `nntp` will pre-fetch the next five articles. If this variable is `t`, `nntp` will pre-fetch all the articles that it can without bound. If it is `nil`, no pre-fetching will be made.

You may wish to create some sort of scheme for choosing which articles that `nntp` should consider as candidates for pre-fetching. For instance, you may wish to pre-fetch all articles with high scores, and not pre-fetch low-scored articles. You can do that by setting the `gnus-asynchronous-article-function`, which will be called with an alist where the keys are the article numbers. Your function should return an alist where the articles you are not interested in have been removed. You could also do sorting on article score and the like.

3.12 Article Caching

If you have an *extremely* slow NNTP connection, you may consider turning article caching on. Each article will then be stored locally under your home directory. As you may surmise, this could potentially use *huge* amounts of disk space, as well as eat up all your inodes so fast it will make your head swim. In vodka.

Used carefully, though, it could be just an easier way to save articles.

To turn caching on, set `gnus-use-cache` to `t`. By default, all articles that are ticked or marked as dormant will then be copied over to your local cache (`gnus-cache-directory`).

Whether this cache is flat or hierarchal is controlled by the `gnus-use-long-file-name` variable, as usual.

When re-select a ticked or dormant article, it will be fetched from the cache instead of from the server. As articles in your cache will never expire, this might serve as a method of saving articles while still keeping them where they belong. Just mark all articles you want to save as dormant, and don't worry.

When an article is marked as read, is it removed from the cache.

The entering/removal of articles from the cache is controlled by the `gnus-cache-enter-articles` and `gnus-cache-remove-articles` variables. Both are lists of symbols. The first is `(ticked dormant)` by default, meaning that ticked and dormant articles will be put in the cache. The latter is `(read)` by default, meaning that articles that are marked as read are removed from the cache. Possibly symbols in these two lists are `ticked`, `dormant`, `unread` and `read`.

So where does the massive article-fetching and storing come into the picture? The `gnus-jog-cache` command will go through all subscribed newsgroups, request all unread articles, and store them in the cache. You should only ever, ever ever ever, use this command if 1) your connection to the NNTP server is really, really, really slow and 2) you have a really, really, really huge disk. Seriously.

It is likely that you do not want caching on some groups. For instance, if your `nnml` mail is located under your home directory, it makes no sense to cache it somewhere else under your home directory. Unless you feel that it's neat to use twice as much space. To limit the caching, you could set the `gnus-uncacheable-groups` regexp to '`^nnml`', for instance. This variable is `nil` by default.

The cache stores information on what articles it contains in its active file (`gnus-cache-active-file`). If this file (or any other parts of the cache) becomes all messed up for some reason or other, Gnus offers two functions that will try to set things right. *`M-x gnus-cache-generate-nov-databases`* will (re)build all the NOV files, and *`gnus-cache-generate-active`* will (re)generate the active file.

3.13 Persistent Articles

Closely related to article caching, we have *persistent articles*. In fact, it's just a different way of looking at caching, and much more useful in my opinion.

Say you're reading a newsgroup, and you happen on to some valuable gem that you want to keep and treasure forever. You'd normally just save it (using one of the many saving commands) in some file. The problem with that is that it's just, well, yucky. Ideally you'd prefer just having the article remain in the group where you found it forever; untouched by the expiry going on at the news server.

This is what a *persistent article* is—an article that just won't be deleted. It's implemented using the normal cache functions, but you use two explicit commands for managing persistent articles:

`*` Make the current article persistent (`gnus-cache-enter-article`).

`M-*` Remove the current article from the persistent articles (`gnus-cache-remove-article`). This will normally delete the article.

Both these commands understand the process/prefix convention.

To avoid having all ticked articles (and stuff) entered into the cache, you should set `gnus-use-cache` to `passive` if you're just interested in persistent articles:

```
(setq gnus-use-cache 'passive)
```

3.14 Article Backlog

If you have a slow connection, but the idea of using caching seems unappealing to you (and it is, really), you can help the situation some by switching on the *backlog*. This is where Gnus will buffer already read articles so that it doesn't have to re-fetch articles you've already read. This only helps if you are in the habit of re-selecting articles you've recently read, of course. If you never do that, turning the backlog on will slow Gnus down a little bit, and increase memory usage some.

If you set `gnus-keep-backlog` to a number *n*, Gnus will store at most *n* old articles in a buffer for later re-fetching. If this variable is non-`nil` and is not a number, Gnus will store *all* read articles, which means that your Emacs will grow without bound before exploding and taking your machine down with you. I put that in there just to keep y'all on your toes.

This variable is `nil` by default.

3.15 Saving Articles

Gnus can save articles in a number of ways. Below is the documentation for saving articles in a fairly straight-forward fashion (i.e., little processing of the article is done before it is saved). For a different approach (uudecoding, unsharing) you should use `gnus-uu` (see Section 3.16 [Decoding Articles], page 51).

If `gnus-save-all-headers` is non-`nil`, Gnus will not delete unwanted headers before saving the article.

If the preceding variable is `nil`, all headers that match the `gnus-saved-headers` regexp will be kept, while the rest will be deleted before saving.

`O o`
`o` Save the current article using the default article saver (`gnus-summary-save-article`).

`O m` Save the current article in mail format (`gnus-summary-save-article-mail`).

`O r` Save the current article in rmail format (`gnus-summary-save-article-rmail`).

`O f` Save the current article in plain file format (`gnus-summary-save-article-file`).

`O b` Save the current article body in plain file format (`gnus-summary-save-article-body-file`).

O h Save the current article in mh folder format (`gnus-summary-save-article-folder`).

O v Save the current article in a VM folder (`gnus-summary-save-article-vm`).

O p Save the current article in a pipe. Uhm, like, what I mean is—Pipe the current article to a process (`gnus-summary-pipe-output`).

All these commands use the process/prefix convention (see Section 8.1 [Process/Prefix], page 119). If you save bunches of articles using these functions, you might get tired of being prompted for files to save each and every article in. The prompting action is controlled by the `gnus-prompt-before-saving` variable, which is `always` by default, giving you that excessive prompting action you know and loathe. If you set this variable to `t` instead, you'll be prompted just once for each series of articles you save. If you like to really have Gnus do all your thinking for you, you can even set this variable to `nil`, which means that you will never be prompted for files to save articles in. Gnus will simply save all the articles in the default files.

You can customize the `gnus-default-article-saver` variable to make Gnus do what you want it to. You can use any of the four ready-made functions below, or you can create your own.

`gnus-summary-save-in-rmail`
: This is the default format, *babyl*. Uses the function in the `gnus-rmail-save-name` variable to get a file name to save the article in. The default is `gnus-plain-save-name`.

`gnus-summary-save-in-mail`
: Save in a Unix mail (mbox) file. Uses the function in the `gnus-mail-save-name` variable to get a file name to save the article in. The default is `gnus-plain-save-name`.

`gnus-summary-save-in-file`
: Append the article straight to an ordinary file. Uses the function in the `gnus-file-save-name` variable to get a file name to save the article in. The default is `gnus-numeric-save-name`.

`gnus-summary-save-body-in-file`
: Append the article body to an ordinary file. Uses the function in the `gnus-file-save-name` variable to get a file name to save the article in. The default is `gnus-numeric-save-name`.

`gnus-summary-save-in-folder`
: Save the article to an MH folder using `rcvstore` from the MH library. Uses the function in the `gnus-folder-save-name` variable to get a file name to save the article in. The default is `gnus-folder-save-name`, but you can also use `gnus-Folder-save-name`. The former creates capitalized names, and the latter does not.

`gnus-summary-save-in-vm`
: Save the article in a VM folder. You have to have the VM mail reader to use this setting.

All of these functions, except for the last one, will save the article in the `gnus-article-save-directory`, which is initialized from the `SAVEDIR` environment variable. This is '`~/News/`' by default.

As you can see above, the functions use different functions to find a suitable name of a file to save the article in. Below is a list of available functions that generate names:

`gnus-Numeric-save-name`
: Generates file names that look like '`~/News/Alt.andrea-dworkin/45`'.

`gnus-numeric-save-name`
: Generates file names that look like '`~/News/alt.andrea-dworkin/45`'.

`gnus-Plain-save-name`
: Generates file names that look like '`~/News/Alt.andrea-dworkin`'.

`gnus-plain-save-name`
: Generates file names that look like '`~/News/alt.andrea-dworkin`'.

You can have Gnus suggest where to save articles by plonking a regexp into the `gnus-split-methods` alist. For instance, if you would like to save articles related to Gnus in the file '`gnus-stuff`', and articles related to VM in `vm-stuff`, you could set this variable to something like:

```
(("^Subject:.*gnus\\|^Newsgroups:.*gnus" "gnus-stuff")
 ("^Subject:.*vm\\|^Xref:.*vm" "vm-stuff")
 (my-choosing-function "../other-dir/my-stuff")
 ((equal gnus-newsgroup-name "mail.misc") "mail-stuff"))
```

We see that this is a list where each element is a list that has two elements—the *match* and the *file*. The match can either be a string (in which case it is used as a regexp to match on the article head); it can be a symbol (which will be called as a function with the group name as a parameter); or it can be a list (which will be `eval`ed). If any of these actions have a non-`nil` result, the *file* will be used as a default prompt. In addition, the result of the operation itself will be used if the function or form called returns a string or a list of strings.

You basically end up with a list of file names that might be used when saving the current article. (All "matches" will be used.) You will then be prompted for what you really want to use as a name, with file name completion over the results from applying this variable.

This variable is `((gnus-article-archive-name))` by default, which means that Gnus will look at the articles it saves for an `Archive-name` line and use that as a suggestion for the file name.

Finally, you have the `gnus-use-long-file-name` variable. If it is `nil`, all the preceding functions will replace all periods ('`.`') in the group names with slashes ('`/`')—which means that the functions will generate hierarchies of directories instead of having all the files in the toplevel directory ('`~/News/alt/andrea-dworkin`' instead of '`~/News/alt.andrea-dworkin`'.) This variable is `t` by default on most systems. However, for historical reasons, this is `nil` on Xenix and usg-unix-v machines by default.

This function also affects kill and score file names. If this variable is a list, and the list contains the element `not-score`, long file names will not be used for score files, if it contains

the element `not-save`, long file names will not be used for saving, and if it contains the element `not-kill`, long file names will not be used for kill files.

If you'd like to save articles in a hierarchy that looks something like a spool, you could

```
(setq gnus-use-long-file-name '(not-save)) ; to get a hierarchy
(setq gnus-default-article-save 'gnus-summary-save-in-file) ; no encoding
```

Then just save with *o*. You'd then read this hierarchy with ephemeral `nneething` groups—*G D* in the group buffer, and the toplevel directory as the argument ("`~/News/`"). Then just walk around to the groups/directories with `nneething`.

3.16 Decoding Articles

Sometime users post articles (or series of articles) that have been encoded in some way or other. Gnus can decode them for you.

All these functions use the process/prefix convention (see Section 8.1 [Process/Prefix], page 119) for finding out what articles to work on, with the extension that a "single article" means "a single series". Gnus can find out by itself what articles belong to a series, decode all the articles and unpack/view/save the resulting file(s).

Gnus guesses what articles are in the series according to the following simplish rule: The subjects must be (nearly) identical, except for the last two numbers of the line. (Spaces are largely ignored, however.)

For example: If you choose a subject called '`cat.gif (2/3)`', Gnus will find all the articles that match the regexp '`^cat.gif ([0-9]+/[0-9]+).*$`'.

Subjects that are nonstandard, like '`cat.gif (2/3) Part 6 of a series`', will not be properly recognized by any of the automatic viewing commands, and you have to mark the articles manually with `#`.

3.16.1 Uuencoded Articles

X u Uudecodes the current series (`gnus-uu-decode-uu`).

X U Uudecodes and saves the current series (`gnus-uu-decode-uu-and-save`).

X v u Uudecodes and views the current series (`gnus-uu-decode-uu-view`).

X v U Uudecodes, views and saves the current series (`gnus-uu-decode-uu-and-save-view`).

Remember that these all react to the presence of articles marked with the process mark. If, for instance, you'd like to decode and save an entire newsgroup, you'd typically do *M P a* (`gnus-uu-mark-all`) and then *X U* (`gnus-uu-decode-uu-and-save`).

All this is very much different from how `gnus-uu` worked with GNUS 4.1, where you had explicit keystrokes for everything under the sun. This version of `gnus-uu` generally assumes that you mark articles in some way (see Section 3.7.5 [Setting Process Marks], page 39) and then press *X u*.

Note: When trying to decode articles that have names matching `gnus-uu-notify-files`, which is hard-coded to '`[Cc][Ii][Nn][Dd][Yy][0-9]+.\\(gif\\|jpg\\)`', `gnus-uu` will automatically post an article on '`comp.unix.wizards`' saying that you have just viewed the file in question. This feature can't be turned off.

3.16.2 Shared Articles

X s Unshars the current series (`gnus-uu-decode-unshar`).

X S Unshars and saves the current series (`gnus-uu-decode-unshar-and-save`).

X v s Unshars and views the current series (`gnus-uu-decode-unshar-view`).

X v S Unshars, views and saves the current series (`gnus-uu-decode-unshar-and-save-view`).

3.16.3 PostScript Files

X p Unpack the current PostScript series (`gnus-uu-decode-postscript`).

X P Unpack and save the current PostScript series (`gnus-uu-decode-postscript-and-save`).

X v p View the current PostScript series (`gnus-uu-decode-postscript-view`).

X v P View and save the current PostScript series (`gnus-uu-decode-postscript-and-save-view`).

3.16.4 Decoding Variables

Adjective, not verb.

3.16.4.1 Rule Variables

Gnus uses *rule variables* to decide how to view a file. All these variables are on the form

```
(list '(regexp1 command2)
      '(regexp2 command2)
      ...)
```

`gnus-uu-user-view-rules`
: This variable is consulted first when viewing files. If you wish to use, for instance, `sox` to convert an '`.au`' sound file, you could say something like:

```
(setq gnus-uu-user-view-rules
  (list '(\"\\\\.au$\" \"sox %s -t .aiff > /dev/audio\")))
```

`gnus-uu-user-view-rules-end`
: This variable is consulted if Gnus couldn't make any matches from the user and default view rules.

`gnus-uu-user-archive-rules`
: This variable can be used to say what commands should be used to unpack archives.

3.16.4.2 Other Decode Variables

`gnus-uu-grabbed-file-functions`
: All functions in this list will be called right each file has been successfully decoded—so that you can move or view files right away, and don't have to wait for all files to be decoded before you can do anything. Ready-made functions you can put in this list are:

  `gnus-uu-grab-view`
  : View the file.

  `gnus-uu-grab-move`
  : Move the file (if you're using a saving function.)

`gnus-uu-ignore-files-by-name`
: Files with name matching this regular expression won't be viewed.

`gnus-uu-ignore-files-by-type`
: Files with a MIME type matching this variable won't be viewed. Note that Gnus tries to guess what type the file is based on the name. `gnus-uu` is not a MIME package (yet), so this is slightly kludgey.

`gnus-uu-tmp-dir`
: Where `gnus-uu` does its work.

`gnus-uu-do-not-unpack-archives`
: Non-`nil` means that `gnus-uu` won't peek inside archives looking for files to display.

`gnus-uu-view-and-save`
: Non-`nil` means that the user will always be asked to save a file after viewing it.

`gnus-uu-ignore-default-view-rules`
: Non-`nil` means that `gnus-uu` will ignore the default viewing rules.

`gnus-uu-ignore-default-archive-rules`
: Non-`nil` means that `gnus-uu` will ignore the default archive unpacking commands.

`gnus-uu-kill-carriage-return`
: Non-`nil` means that `gnus-uu` will strip all carriage returns from articles.

`gnus-uu-unmark-articles-not-decoded`
: Non-`nil` means that `gnus-uu` will mark articles that were unsuccessfully decoded as unread.

`gnus-uu-correct-stripped-uucode`
: Non-`nil` means that `gnus-uu` will *try* to fix uuencoded files that have had trailing spaces deleted.

`gnus-uu-view-with-metamail`
: Non-`nil` means that `gnus-uu` will ignore the viewing commands defined by the rule variables and just fudge a MIME content type based on the file name. The result will be fed to `metamail` for viewing.

`gnus-uu-save-in-digest`
: Non-`nil` means that `gnus-uu`, when asked to save without decoding, will save in digests. If this variable is `nil`, `gnus-uu` will just save everything in a file without any embellishments. The digesting almost conforms to RFC1153—no easy way to specify any meaningful volume and issue numbers were found, so I simply dropped them.

3.16.4.3 Uuencoding and Posting

`gnus-uu-post-include-before-composing`
: Non-`nil` means that `gnus-uu` will ask for a file to encode before you compose the article. If this variable is `t`, you can either include an encoded file with `C-c C-i` or have one included for you when you post the article.

`gnus-uu-post-length`
: Maximum length of an article. The encoded file will be split into how many articles it takes to post the entire file.

`gnus-uu-post-threaded`
: Non-`nil` means that `gnus-uu` will post the encoded file in a thread. This may not be smart, as no other decoder I have seen are able to follow threads when collecting uuencoded articles. (Well, I have seen one package that does that—`gnus-uu`, but somehow, I don't think that counts...) Default is `nil`.

`gnus-uu-post-separate-description`
: Non-`nil` means that the description will be posted in a separate article. The first article will typically be numbered (0/x). If this variable is `nil`, the description the user enters will be included at the beginning of the first article, which will be numbered (1/x). Default is `t`.

3.16.5 Viewing Files

After decoding, if the file is some sort of archive, Gnus will attempt to unpack the archive and see if any of the files in the archive can be viewed. For instance, if you have a gzipped tar file '`pics.tar.gz`' containing the files '`pic1.jpg`' and '`pic2.gif`', Gnus will uncompress and de-tar the main file, and then view the two pictures. This unpacking process is recursive, so if the archive contains archives of archives, it'll all be unpacked.

Finally, Gnus will normally insert a *pseudo-article* for each extracted file into the summary buffer. If you go to these "articles", you will be prompted for a command to run (usually Gnus will make a suggestion), and then the command will be run.

If `gnus-view-pseudo-asynchronously` is `nil`, Emacs will wait until the viewing is done before proceeding.

If `gnus-view-pseudos` is `automatic`, Gnus will not insert the pseudo-articles into the summary buffer, but view them immediately. If this variable is `not-confirm`, the user won't even be asked for a confirmation before viewing is done.

If `gnus-view-pseudos-separately` is non-`nil`, one pseudo-article will be created for each file to be viewed. If `nil`, all files that use the same viewing command will be given as a list of parameters to that command.

If `gnus-insert-pseudo-articles` is non-`nil`, insert pseudo-articles when decoding. It is `t` by default.

So; there you are, reading your *pseudo-articles* in your *virtual newsgroup* from the *virtual server*; and you think: Why isn't anything real anymore? How did we get here?

3.17 Article Treatment

Reading through this huge manual, you may have quite forgotten that the object of newsreaders are to actually, like, read what people have written. Reading articles. Unfortunately, people are quite bad at writing, so there are tons of functions and variables to make reading these articles easier.

3.17.1 Article Highlighting

Not only do you want your article buffer to look like fruit salad, but you want it to look like technicolor fruit salad.

W H `a` Highlight the current article (`gnus-article-highlight`).

W H `h` Highlight the headers (`gnus-article-highlight-headers`). The highlighting will be done according to the `gnus-header-face-alist` variable, which is a list where each element has the form *(regexp name content)*. *regexp* is a regular expression for matching the header, *name* is the face used for highlighting the header name and *content* is the face for highlighting the header value. The first match made will be used. Note that *regexp* shouldn't have '`^`' prepended—Gnus will add one.

W H `c` Highlight cited text (`gnus-article-highlight-citation`).

Some variables to customize the citation highlights:

`gnus-cite-parse-max-size`
: If the article size if bigger than this variable (which is 25000 by default), no citation highlighting will be performed.

`gnus-cite-prefix-regexp`
: Regexp matching the longest possible citation prefix on a line.

`gnus-cite-max-prefix`
: Maximum possible length for a citation prefix (default 20).

`gnus-cite-face-list`
: List of faces used for highlighting citations. When there are citations from multiple articles in the same message, Gnus will try to give each citation from each article its own face. This should make it easier to see who wrote what.

`gnus-supercite-regexp`
: Regexp matching normal Supercite attribution lines.

`gnus-supercite-secondary-regexp`
: Regexp matching mangled Supercite attribution lines.

`gnus-cite-minimum-match-count`
: Minimum number of identical prefixes we have to see before we believe that it's a citation.

`gnus-cite-attribution-prefix`
: Regexp matching the beginning of an attribution line.

`gnus-cite-attribution-suffix`
: Regexp matching the end of an attribution line.

`gnus-cite-attribution-face`
: Face used for attribution lines. It is merged with the face for the cited text belonging to the attribution.

`W H s` Highlight the signature (`gnus-article-highlight-signature`). Everything after `gnus-signature-separator` in an article will be considered a signature and will be highlighted with `gnus-signature-face`, which is `italic` by default.

3.17.2 Article Hiding

Or rather, hiding certain things in each article. There usually is much too much cruft in most articles.

`W W a` Do maximum hiding on the summary buffer (*`gnus-article-hide`*).

`W W h` Hide headers (`gnus-article-hide-headers`). See Section 4.1 [Hiding Headers], page 69.

`W W b` Hide headers that aren't particularly interesting (`gnus-article-hide-boring-headers`). See Section 4.1 [Hiding Headers], page 69.

`W W s` Hide signature (`gnus-article-hide-signature`).

`W W p` Hide PGP signatures (`gnus-article-hide-pgp`).

W W c Hide citation (`gnus-article-hide-citation`). Some variables for customizing the hiding:

`gnus-cite-hide-percentage`
: If the cited text is of a bigger percentage than this variable (default 50), hide the cited text.

`gnus-cite-hide-absolute`
: The cited text must be have at least this length (default 10) before it is hidden.

`gnus-cited-text-button-line-format`
: Gnus adds buttons show where the cited text has been hidden, and to allow toggle hiding the text. The format of the variable is specified by this format-like variable. These specs are legal:

  ‘`b`’ Start point of the hidden text.

  ‘`e`’ End point of the hidden text.

  ‘`l`’ Length of the hidden text.

`gnus-cited-lines-visible`
: The number of lines at the beginning of the cited text to leave shown.

W W C Hide cited text in articles that aren't roots (`gnus-article-hide-citation-in-followups`). This isn't very useful as an interactive command, but might be a handy function to stick in `gnus-article-display-hook` (see Section 4.3 [Customizing Articles], page 71).

All these “hiding” commands are toggles, but if you give a negative prefix to these commands, they will show what they have previously hidden. If you give a positive prefix, they will always hide.

Also see Section 3.17.1 [Article Highlighting], page 55 for further variables for citation customization.

`gnus-signature-limit` provides a limit to what is considered a signature. If it is a number, no signature may not be longer (in characters) than that number. If it is a function, the function will be called without any parameters, and if it returns `nil`, there is no signature in the buffer. If it is a string, it will be used as a regexp. If it matches, the text in question is not a signature.

3.17.3 Article Washing

We call this “article washing” for a really good reason. Namely, the *A* key was taken, so we had to use the *W* key instead.

Washing is defined by us as “changing something from something to something else”, but normally results in something looking better. Cleaner, perhaps.

W l Remove page breaks from the current article (`gnus-summary-stop-page-breaking`).

`W r` Do a Caesar rotate (rot13) on the article buffer (`gnus-summary-caesar-message`).

`W t` Toggle whether to display all headers in the article buffer (`gnus-summary-toggle-header`).

`W v` Toggle whether to display all headers in the article buffer permanently (`gnus-summary-verbose-header`).

`W m` Toggle whether to run the article through MIME before displaying (`gnus-summary-toggle-mime`).

`W o` Treat overstrike (`gnus-article-treat-overstrike`).

`W w` Do word wrap (`gnus-article-fill-cited-article`).

`W c` Remove CR (`gnus-article-remove-cr`).

`W L` Remove all blank lines at the end of the article (`gnus-article-remove-trailing-blank-lines`).

`W q` Treat quoted-printable (`gnus-article-de-quoted-unreadable`).

`W f` Look for and display any X-Face headers (`gnus-article-display-x-face`). The command executed by this function is given by the `gnus-article-x-face-command` variable. If this variable is a string, this string will be executed in a sub-shell. If it is a function, this function will be called with the face as the argument. If the `gnus-article-x-face-too-ugly` (which is a regexp) matches the `From` header, the face will not be shown. The default action under Emacs is to fork off an `xv` to view the face; under XEmacs the default action is to display the face before the `From` header. (It's nicer if XEmacs has been compiled with X-Face support – that will make display somewhat faster. If there's no native X-Face support, Gnus will try to convert the `X-Face` header using external programs from the `pbmplus` package and friends.) If you want to have this function in the display hook, it should probably come last.

`W b` Add clickable buttons to the article (`gnus-article-add-buttons`).

`W B` Add clickable buttons to the article headers (`gnus-article-add-buttons-to-head`).

3.17.4 Article Buttons

People often include references to other stuff in articles, and it would be nice if Gnus could just fetch whatever it is that people talk about with the minimum of fuzz.

Gnus adds *buttons* to certain standard references by default: Well-formed URLs, mail addresses and Message-IDs. This is controlled by two variables, one that handles article bodies and one that handles article heads:

`gnus-button-alist`
: This is an alist where each entry has this form:

```
(REGEXP BUTTON-PAR USE-P FUNCTION DATA-PAR)
```

regexp All text that match this regular expression will be considered an external reference. Here's a typical regexp that match embedded URLs: '`<URL:\\([^\n\r>]*\\)>`'.

button-par
Gnus has to know which parts of the match is to be highlighted. This is a number that says what sub-expression of the regexp that is to be highlighted. If you want it all highlighted, you use `0` here.

use-p This form will be `eval`ed, and if the result is non-`nil`, this is considered a match. This is useful if you want extra sifting to avoid false matches.

function This function will be called when you click on this button.

data-par As with *button-par*, this is a sub-expression number, but this one says which part of the match is to be sent as data to *function*.

So the full entry for buttonizing URLs is then

```
("<URL:\\([^\n\r>]*\\)>" 0 t gnus-button-url 1)
```

`gnus-header-button-alist`
This is just like the other alist, except that it is applied to the article head only, and that each entry has an additional element that is used to say what headers to apply the buttonize coding to:

```
(HEADER REGEXP BUTTON-PAR USE-P FUNCTION DATA-PAR)
```

header is a regular expression.

`gnus-button-url-regexp`
A regular expression that matches embedded URLs. It is used in the default values of the variables above.

`gnus-article-button-face`
Face used on bottons.

`gnus-article-mouse-face`
Face is used when the mouse cursor is over a button.

3.17.5 Article Date

The date is most likely generated in some obscure timezone you've never heard of, so it's quite nice to be able to find out what the time was when the article was sent.

W T u Display the date in UT (aka. GMT, aka ZULU) (`gnus-article-date-ut`).

W T l Display the date in the local timezone (`gnus-article-date-local`).

W T e Say how much time has (e)lapsed between the article was posted and now (`gnus-article-date-lapsed`).

W T o Display the original date (`gnus-article-date-original`). This can be useful if you normally use some other conversion function and is worried that it might

be doing something totally wrong. Say, claiming that the article was posted in 1854. Although something like that is *totally* impossible. Don't you trust me? *titter*

3.18 Summary Sorting

You can have the summary buffer sorted in various ways, even though I can't really see why you'd want that.

C-c C-s C-n
: Sort by article number (`gnus-summary-sort-by-number`).

C-c C-s C-a
: Sort by author (`gnus-summary-sort-by-author`).

C-c C-s C-s
: Sort by subject (`gnus-summary-sort-by-subject`).

C-c C-s C-d
: Sort by date (`gnus-summary-sort-by-date`).

C-c C-s C-i
: Sort by score (`gnus-summary-sort-by-score`).

These functions will work both when you use threading and when you don't use threading. In the latter case, all summary lines will be sorted, line by line. In the former case, sorting will be done on a root-by-root basis, which might not be what you were looking for. To toggle whether to use threading, type *T T* (see Section 3.9.2 [Thread Commands], page 43).

3.19 Finding the Parent

If you'd like to read the parent of the current article, and it is not displayed in the summary buffer, you might still be able to. That is, if the current group is fetched by NNTP, the parent hasn't expired and the `References` in the current article are not mangled, you can just press `^` or *A r* (`gnus-summary-refer-parent-article`). If everything goes well, you'll get the parent. If the parent is already displayed in the summary buffer, point will just move to this article.

You can have Gnus fetch all articles mentioned in the `References` header of the article by pushing *A R* (`gnus-summary-refer-references`).

You can also ask the NNTP server for an arbitrary article, no matter what group it belongs to. *M-^* (`gnus-summary-refer-article`) will ask you for a `Message-ID`, which is one of those long thingies that look something like '`<38o6up$6f2@hymir.ifi.uio.no>`'. You have to get it all exactly right. No fuzzy searches, I'm afraid.

If the group you are reading is located on a backend that does not support fetching by `Message-ID` very well (like `nnspool`), you can set `gnus-refer-article-method` to an

NNTP method. It would, perhaps, be best if the NNTP server you consult is the same as the one that keeps the spool you are reading from updated, but that's not really necessary.

Most of the mail backends support fetching by `Message-ID`, but do not do a particularly excellent job of it. That is, `nnmbox` and `nnbabyl` are able to locate articles from any groups, while `nnml` and `nnfolder` are only able to locate articles that have been posted to the current group. (Anything else would be too time consuming.) `nnmh` does not support this at all.

3.20 Alternative Approaches

Different people like to read news using different methods. This being Gnus, we offer a small selection of minor modes for the summary buffers.

3.20.1 Pick and Read

Some newsreaders (like `nn` and, uhm, `nn`) use a two-phased reading interface. The user first marks the articles she wants to read from a summary buffer. Then she starts reading the articles with just an article buffer displayed.

Gnus provides a summary buffer minor mode that allows this—`gnus-pick-mode`. This basically means that a few process mark commands become one-keystroke commands to allow easy marking, and it makes one additional command for switching to the summary buffer available.

Here are the available keystrokes when using pick mode:

SPACE Pick the article (`gnus-summary-mark-as-processable`).

u Unpick the article (`gnus-summary-unmark-as-processable`).

U Unpick all articles (`gnus-summary-unmark-all-processable`).

t Pick the thread (`gnus-uu-mark-thread`).

T Unpick the thread (`gnus-uu-unmark-thread`).

r Pick the region (`gnus-uu-mark-region`).

R Unpick the region (`gnus-uu-unmark-region`).

e Pick articles that match a regexp (`gnus-uu-mark-by-regexp`).

E Unpick articles that match a regexp (`gnus-uu-unmark-by-regexp`).

b Pick the buffer (`gnus-uu-mark-buffer`).

B Unpick the buffer (`gnus-uu-unmark-buffer`).

RET Start reading the picked articles (`gnus-pick-start-reading`). If given a prefix, mark all unpicked articles as read first. If `gnus-pick-display-summary` is non-`nil`, the summary buffer will still be visible when you are reading.

If this sounds like a good idea to you, you could say:

```
(add-hook 'gnus-summary-mode-hook 'gnus-pick-mode)
```

`gnus-pick-mode-hook` is run in pick minor mode buffers.

3.20.2 Binary Groups

If you spend much time in binary groups, you may grow tired of hitting *X u*, *n*, *RET* all the time. *M-x gnus-binary-mode* is a minor mode for summary buffers that makes all ordinary Gnus article selection functions uudecode series of articles and display the result instead of just displaying the articles the normal way.

In fact, the only way to see the actual articles if you have turned this mode on is the *g* command (`gnus-binary-show-article`).

`gnus-binary-mode-hook` is called in binary minor mode buffers.

3.21 Tree Display

If you don't like the normal Gnus summary display, you might try setting `gnus-use-trees` to `t`. This will create (by default) an additional *tree buffer*. You can execute all summary mode commands in the tree buffer.

There are a few variables to customize the tree display, of course:

`gnus-tree-mode-hook`
: A hook called in all tree mode buffers.

`gnus-tree-mode-line-format`
: A format string for the mode bar in the tree mode buffers. The default is '`Gnus: %%b [%A] %Z`'. For a list of legal specs, see Section 3.1.2 [Summary Buffer Mode Line], page 30.

`gnus-selected-tree-face`
: Face used for highlighting the selected article in the tree buffer. The default is `modeline`.

`gnus-tree-line-format`
: A format string for the tree nodes. The name is a bit of a misnomer, though—it doesn't define a line, but just the node. The default value is '`%(%[%3,3n%]%)`', which displays the first three characters of the name of the poster. It is vital that all nodes are of the same length, so you *must* use '`%4,4n`'-like specifiers.

  Legal specs are:

  '`n`' The name of the poster.

  '`f`' The `From` header.

  '`N`' The number of the article.

  '`[`' The opening bracket.

  '`]`' The closing bracket.

  '`s`' The subject.

  See Section 8.3 [Formatting Variables], page 120.

  Variables related to the display are:

`gnus-tree-brackets`
: This is used for differentiating between "real" articles and "sparse" articles. The format is *((real-open . real-close) (sparse-open . sparse-close) (dummy-open . dummy-close))*, and the default is `((?[ . ?]) (?( . ?)) (?{ . ?}))`.

`gnus-tree-parent-child-edges`
: This is a list that contains the characters used for connecting parent nodes to their children. The default is `(?- ?\\ ?|)`.

`gnus-tree-minimize-window`
: If this variable is non-`nil`, Gnus will try to keep the tree buffer as small as possible to allow more room for the other Gnus windows. If this variable is a number, the tree buffer will never be higher than that number. The default is `t`.

`gnus-generate-tree-function`
: The function that actually generates the thread tree. Two predefined functions are available: `gnus-generate-horizontal-tree` and `gnus-generate-vertical-tree` (which is the default).

Here's and example from a horizontal tree buffer:

```
{***}-(***)-[odd]-[Gun]
      |     \[Jan]
      |     \[odd]-[Eri]
      |     \(***)-[Eri]
      |           \[odd]-[Paa]
      \[Bjo]
      \[Gun]
      \[Gun]-[Jor]
```

Here's the same thread displayed in a vertical tree buffer:

```
{***}
  |-------------------------\-----\-----\
(***)                       [Bjo] [Gun] [Gun]
  |--\-----\-----\                        |
[odd] [Jan] [odd] (***)                 [Jor]
  |           |     |--\
[Gun]       [Eri] [Eri] [odd]
                        |
                      [Paa]
```

3.22 Mail Group Commands

Some commands only make sense in mail groups. If these commands are illegal in the current group, they will raise a hell and let you know.

All these commands (except the expiry and edit commands) use the process/prefix convention (see Section 8.1 [Process/Prefix], page 119).

B e Expire all expirable articles in the group (`gnus-summary-expire-articles`).

B M-C-e Expunge all the expirable articles in the group (`gnus-summary-expire-articles-now`). This means that **all** articles that are eligible for expiry in the current group will disappear forever into that big '`/dev/null`' in the sky.

B DEL Delete the mail article. This is "delete" as in "delete it from your disk forever and ever, never to return again." Use with caution. (`gnus-summary-delete-article`).

B m Move the article from one mail group to another (`gnus-summary-move-article`).

B c Copy the article from one group (mail group or not) to a mail group (`gnus-summary-copy-article`).

B C Crosspost the current article to some other group (`gnus-summary-crosspost-article`). This will create a new copy of the article in the other group, and the Xref headers of the article will be properly updated.

B i Import an arbitrary file into the current mail newsgroup (`gnus-summary-import-article`). You will be prompted for a file name, a `From` header and a `Subject` header.

B r Respool the mail article (`gnus-summary-move-article`).

B w
e Edit the current article (`gnus-summary-edit-article`). To finish editing and make the changes permanent, type *C-c C-c* (*`gnus-summary-edit-article-done`*).

B q If you want to re-spool an article, you might be curious as to what group the article will end up in before you do the re-spooling. This command will tell you (`gnus-summary-respool-query`).

If you move (or copy) articles regularly, you might wish to have Gnus suggest where to put the articles. `gnus-move-split-methods` is a variable that uses the same syntax as `gnus-split-methods` (see Section 3.15 [Saving Articles], page 48). You may customize that variable to create suggestions you find reasonable.

3.23 Various Summary Stuff

`gnus-summary-mode-hook`
This hook is called when creating a summary mode buffer.

`gnus-summary-generate-hook`
This is called as the last thing before doing the threading and the generation of the summary buffer. It's quite convenient for customizing the threading variables based on what data the newsgroup has. This hook is called from the summary buffer after most summary buffer variables has been set.

`gnus-summary-prepare-hook`
: Is is called after the summary buffer has been generated. You might use it to, for instance, highlight lines or modify the look of the buffer in some other ungodly manner. I don't care.

3.23.1 Summary Group Information

H f
: Try to fetch the FAQ (list of frequently asked questions) for the current group (`gnus-summary-fetch-faq`). Gnus will try to get the FAQ from `gnus-group-faq-directory`, which is usually a directory on a remote machine. This variable can also be a list of directories. In that case, giving a prefix to this command will allow you to choose between the various sites. `ange-ftp` probably will be used for fetching the file.

H d
: Give a brief description of the current group (`gnus-summary-describe-group`). If given a prefix, force rereading the description from the server.

H h
: Give a very brief description of the most important summary keystrokes (`gnus-summary-describe-briefly`).

H i
: Go to the Gnus info node (`gnus-info-find-node`).

3.23.2 Searching for Articles

M-s
: Search through all subsequent articles for a regexp (`gnus-summary-search-article-forward`).

M-r
: Search through all previous articles for a regexp (`gnus-summary-search-article-backward`).

&
: This command will prompt you for a header field, a regular expression to match on this field, and a command to be executed if the match is made (`gnus-summary-execute-command`).

M-&
: Perform any operation on all articles that have been marked with the process mark (`gnus-summary-universal-argument`).

3.23.3 Really Various Summary Commands

A D
: If the current article is a collection of other articles (for instance, a digest), you might use this command to enter a group based on the that article (`gnus-summary-enter-digest-group`). Gnus will try to guess what article type is currently displayed unless you give a prefix to this command, which forces a "digest" interpretation. Basically, whenever you see a message that is a collection of other messages on some format, you *A D* and read these messages in a more convenient fashion.

`C-t` Toggle truncation of summary lines (`gnus-summary-toggle-truncation`).

`=` Expand the summary buffer window (`gnus-summary-expand-window`). If given a prefix, force an `article` window configuration.

3.24 Exiting the Summary Buffer

Exiting from the summary buffer will normally update all info on the group and return you to the group buffer.

`Z Z`
`q` Exit the current group and update all information on the group (`gnus-summary-exit`). `gnus-summary-prepare-exit-hook` is called before doing much of the exiting, and calls `gnus-summary-expire-articles` by default. `gnus-summary-exit-hook` is called after finishing the exiting process.

`Z E`
`Q` Exit the current group without updating any information on the group (`gnus-summary-exit-no-update`).

`Z c`
`c` Mark all unticked articles in the group as read and then exit (`gnus-summary-catchup-and-exit`).

`Z C` Mark all articles, even the ticked ones, as read and then exit (`gnus-summary-catchup-all-and-exit`).

`Z n` Mark all articles as read and go to the next group (`gnus-summary-catchup-and-goto-next-group`).

`Z R` Exit this group, and then enter it again (`gnus-summary-reselect-current-group`). If given a prefix, select all articles, both read and unread.

`Z G`
`M-g` Exit the group, check for new articles in the group, and select the group (`gnus-summary-rescan-group`). If given a prefix, select all articles, both read and unread.

`Z N` Exit the group and go to the next group (`gnus-summary-next-group`).

`Z P` Exit the group and go to the previous group (`gnus-summary-prev-group`).

`gnus-exit-group-hook` is called when you exit the current group.

If you're in the habit of exiting groups, and then changing your mind about it, you might set `gnus-kill-summary-on-exit` to `nil`. If you do that, Gnus won't kill the summary buffer when you exit it. (Quelle surprise!) Instead it will change the name of the buffer to something like '`*Dead Summary ... *`' and install a minor mode called `gnus-dead-summary-mode`. Now, if you switch back to this buffer, you'll find that all keys are mapped to a function called `gnus-summary-wake-up-the-dead`. So tapping any keys in a dead summary buffer will result in a live, normal summary buffer.

There will never be more than one dead summary buffer at any one time.

The data on the current group will be updated (which articles you have read, which articles you have replied to, etc.) when you exit the summary buffer. If the `gnus-use-cross-reference` variable is `t` (which is the default), articles that are cross-referenced to this group and are marked as read, will also be marked as read in the other subscribed groups they were cross-posted to. If this variable is neither `nil` nor `t`, the article will be marked as read in both subscribed and unsubscribed groups.

Marking cross-posted articles as read ensures that you'll never have to read the same article more than once. Unless, of course, somebody has posted it to several groups separately. Posting the same article to several groups (not cross-posting) is called *spamming*, and you are by law required to send nasty-grams to anyone who perpetrates such a heinous crime.

Remember: Cross-posting is kinda ok, but posting the same article separately to several groups is not. Massive cross-posting (aka. *velveeta*) is to be avoided.

One thing that may cause Gnus to not do the cross-posting thing correctly is if you use an NNTP server that supports XOVER (which is very nice, because it speeds things up considerably) which does not include the `Xref` header in its NOV lines. This is Evil, but all too common, alas, alack. Gnus tries to Do The Right Thing even with XOVER by registering the `Xref` lines of all articles you actually read, but if you kill the articles, or just mark them as read without reading them, Gnus will not get a chance to snoop the `Xref` lines out of these articles, and will be unable to use the cross reference mechanism.

To check whether your NNTP server includes the `Xref` header in its overview files, try '`telnet your.nntp.server nntp`', '`MODE READER`' on `inn` servers, and then say '`LIST overview.fmt`'. This may not work, but if it does, and the last line you get does not read '`Xref:full`', then you should shout and whine at your news admin until she includes the `Xref` header in the overview files.

If you want Gnus to get the `Xref`s right all the time, you have to set `gnus-nov-is-evil` to `t`, which slows things down considerably.

C'est la vie.

4 The Article Buffer

The articles are displayed in the article buffer, of which there is only one. All the summary buffers share the same article buffer unless you tell Gnus otherwise.

4.1 Hiding Headers

The top section of each article is the *head*. (The rest is the *body*, but you may have guessed that already.)

There is a lot of useful information in the head: the name of the person who wrote the article, the date it was written and the subject of the article. That's well and nice, but there's also lots of information most people do not want to see—what systems the article has passed through before reaching you, the `Message-ID`, the `References`, etc. ad nauseum—and you'll probably want to get rid of some of those lines. If you want to keep all those lines in the article buffer, you can set `gnus-show-all-headers` to `t`.

Gnus provides you with two variables for sifting headers:

`gnus-visible-headers`
: If this variable is non-`nil`, it should be a regular expression that says what headers you wish to keep in the article buffer. All headers that do not match this variable will be hidden.

  For instance, if you only want to see the name of the person who wrote the article and the subject, you'd say:

  ```
  (setq gnus-visible-headers "^From:\\|^Subject:")
  ```

  This variable can also be a list of regexps to match headers that are to remain visible.

`gnus-ignored-headers`
: This variable is the reverse of `gnus-visible-headers`. If this variable is set (and `gnus-visible-headers` is `nil`), it should be a regular expression that matches all lines that you want to hide. All lines that do not match this variable will remain visible.

  For instance, if you just want to get rid of the `References` line and the `Xref` line, you might say:

  ```
  (setq gnus-ignored-headers "^References:\\|^Xref:")
  ```

  This variable can also be a list of regexps to match headers that are to be removed.

  Note that if `gnus-visible-headers` is non-`nil`, this variable will have no effect.

Gnus can also sort the headers for you. (It does this by default.) You can control the sorting by setting the `gnus-sorted-header-list` variable. It is a list of regular expressions that says in what order the headers are to be displayed.

For instance, if you want the name of the author of the article first, and then the subject, you might say something like:

```
(setq gnus-sorted-header-list '("^From:" "^Subject:"))
```

Any headers that are to remain visible, but are not listed in this variable, will be displayed in random order after all the headers that are listed in this variable.

You can hide further boring headers by entering `gnus-article-hide-boring-headers` into `gnus-article-display-hook`. What this function does depends on the `gnus-boring-article-headers` variable. It's a list, but this list doesn't actually contain header names. Instead is lists various *boring conditions* that Gnus can check and remove from sight.

These conditions are:

`empty` Remove all empty headers.

`newsgroups`
: Remove the `Newsgroups` header if it only contains the current group name.

`followup-to`
: Remove the `Followup-To` header if it is identical to the `Newsgroups` header.

`reply-to` Remove the `Reply-To` header if it lists the same address as the `From` header.

`date` Remove the `Date` header if the article is less than three days old.

To include the four first elements, you could say something like;

```
(setq gnus-boring-article-headers
      '(empty newsgroups followup-to reply-to))
```

This is also the default value for this variable.

4.2 Using MIME

Mime is a standard for waving your hands through the air, aimlessly, while people stand around yawning.

MIME, however, is a standard for encoding your articles, aimlessly, while all newsreaders die of fear.

MIME may specify what character set the article uses, the encoding of the characters, and it also makes it possible to embed pictures and other naughty stuff in innocent-looking articles.

Gnus handles MIME by shoving the articles through `gnus-show-mime-method`, which is `metamail-buffer` by default. Set `gnus-show-mime` to `t` if you want to use MIME all the time. However, if `gnus-strict-mime` is non-`nil`, the MIME method will only be used if there are MIME headers in the article.

It might be best to just use the toggling functions from the summary buffer to avoid getting nasty surprises. (For instance, you enter the group '`alt.sing-a-long`' and, before you know it, MIME has decoded the sound file in the article and some horrible sing-a-long song comes streaming out out your speakers, and you can't find the volume button, because there isn't one, and people are starting to look at you, and you try to stop the program, but you can't, and you can't find the program to control the volume, and everybody else in the room suddenly decides to look at you disdainfully, and you'll feel rather stupid.)

Any similarity to real events and people is purely coincidental. Ahem.

4.3 Customizing Articles

The `gnus-article-display-hook` is called after the article has been inserted into the article buffer. It is meant to handle all treatment of the article before it is displayed.

By default it contains `gnus-article-hide-headers`, `gnus-article-treat-overstrike`, and `gnus-article-maybe-highlight`, but there are thousands, nay millions, of functions you can put in this hook. For an overview of functions see Section 3.17.1 [Article Highlighting], page 55, see Section 3.17.2 [Article Hiding], page 56, see Section 3.17.3 [Article Washing], page 57, see Section 3.17.4 [Article Buttons], page 58 and see Section 3.17.5 [Article Date], page 59.

You can, of course, write your own functions. The functions are called from the article buffer, and you can do anything you like, pretty much. There is no information that you have to keep in the buffer—you can change everything. However, you shouldn't delete any headers. Instead make them invisible if you want to make them go away.

4.4 Article Keymap

Most of the keystrokes in the summary buffer can also be used in the article buffer. They should behave as if you typed them in the summary buffer, which means that you don't actually have to have a summary buffer displayed while reading. You can do it all from the article buffer.

A few additional keystrokes are available:

SPACE Scroll forwards one page (`gnus-article-next-page`).

DEL Scroll backwards one page (`gnus-article-prev-page`).

C-c ^ If point is in the neighborhood of a `Message-ID` and you press *r*, Gnus will try to get that article from the server (`gnus-article-refer-article`).

C-c C-m Send a reply to the address near point (`gnus-article-mail`). If given a prefix, include the mail.

s Reconfigure the buffers so that the summary buffer becomes visible (`gnus-article-show-summary`).

? Give a very brief description of the available keystrokes (`gnus-article-describe-briefly`).

TAB Go to the next button, if any (`gnus-article-next-button`. This only makes sense if you have buttonizing turned on.

M-TAB Go to the previous button, if any (`gnus-article-prev-button`.

4.5 Misc Article

`gnus-single-article-buffer`
: If non-`nil`, use the same article buffer for all the groups. (This is the default.) If `nil`, each group will have its own article buffer.

`gnus-article-prepare-hook`
: This hook is called right after the article has been inserted into the article buffer. It is mainly intended for functions that do something depending on the contents; it should probably not be used for changing the contents of the article buffer.

`gnus-article-display-hook`
: This hook is called as the last thing when displaying an article, and is intended for modifying the contents of the buffer, doing highlights, hiding headers, and the like.

`gnus-article-mode-hook`
: Hook called in article mode buffers.

`gnus-article-mode-line-format`
: This variable is a format string along the same lines as `gnus-summary-mode-line-format`. It accepts exactly the same format specifications as that variable.

`gnus-break-pages`
: Controls whether *page breaking* is to take place. If this variable is non-`nil`, the articles will be divided into pages whenever a page delimiter appears in the article. If this variable is `nil`, paging will not be done.

`gnus-page-delimiter`
: This is the delimiter mentioned above. By default, it is '`^L`' (form linefeed).

5 Composing Messages

All commands for posting and mailing will put you in a message buffer where you can edit the article all you like, before you send the article by pressing `C-c C-c`. See section "Top" in *The Message Manual*. If you are in a foreign news group, and you wish to post the article using the foreign server, you can give a prefix to `C-c C-c` to make Gnus try to post using the foreign server.

Also see see Section 3.6 [Canceling and Superseding], page 36 for information on how to remove articles you shouldn't have posted.

5.1 Mail

Variables for customizing outgoing mail:

`gnus-uu-digest-headers`
: List of regexps to match headers included in digested messages. The headers will be included in the sequence they are matched.

5.2 Post

Variables for composing news articles:

`gnus-sent-message-ids-file`
: Gnus will keep a `Message-ID` history file of all the mails it has sent. If it discovers that it has already sent a mail, it will ask the user whether to re-send the mail. (This is primarily useful when dealing with SOUP packets and the like where one is apt to sent the same packet multiple times.) This variable says what the name of this history file is. It is '`~/News/Sent-Message-IDs`' by default. Set this variable to `nil` if you don't want Gnus to keep a history file.

`gnus-sent-message-ids-length`
: This variable says how many `Message-ID`s to keep in the history file. It is 1000 by default.

5.3 Posting Server

When you press those magical `C-c C-c` keys to ship off your latest (extremely intelligent, of course) article, where does it go?

Thank you for asking. I hate you.

It can be quite complicated. Normally, Gnus will use the same native server. However. If your native server doesn't allow posting, just reading, you probably want to use some other server to post your (extremely intelligent and fabulously interesting) articles. You can then set the `gnus-post-method` to some other method:

```
(setq gnus-post-method '(nnspool ""))
```

Now, if you've done this, and then this server rejects your article, or this server is down, what do you do then? To override this variable you can use a non-zero prefix to the `C-c C-c` command to force using the "current" server for posting.

If you give a zero prefix (i. e., `C-u 0 C-c C-c`) to that command, Gnus will prompt you for what method to use for posting.

You can also set `gnus-post-method` to a list of select methods. If that's the case, Gnus will always prompt you for what method to use for posting.

5.4 Mail and Post

Here's a list of variables that are relevant to both mailing and posting:

`gnus-mailing-list-groups`
: If your news server offers groups that are really mailing lists that are gatewayed to the NNTP server, you can read those groups without problems, but you can't post/followup to them without some difficulty. One solution is to add a `to-address` to the group parameters (see Section 2.9 [Group Parameters], page 19). An easier thing to do is set the `gnus-mailing-list-groups` to a regexp that match the groups that really are mailing lists. Then, at least, followups to the mailing lists will work most of the time. Posting to these groups (`a`) is still a pain, though.

You may want to do spell-checking on messages that you send out. Or, if you don't want to spell-check by hand, you could add automatic spell-checking via the `ispell` package:

```
(add-hook 'message-send-hook 'ispell-message)
```

5.5 Archived Messages

Gnus provides a few different methods for storing the mail you send. The default method is to use the *archive virtual server* to store the mail. If you want to disable this completely, you should set `gnus-message-archive-group` to `nil`.

`gnus-message-archive-method` says what virtual server Gnus is to use to store sent messages. It is `(nnfolder "archive" (nnfolder-directory "~/Mail/archive/"))` by default, but you can use any mail select method (`nnml`, `nnmbox`, etc.). However, `nnfolder` is a quite likeable select method for doing this sort of thing. If you don't like the default directory chosen, you could say something like:

```
(setq gnus-message-archive-method
      '(nnfolder "archive"
                 (nnfolder-inhibit-expiry t)
                 (nnfolder-active-file "~/News/sent-mail/active")
                 (nnfolder-directory "~/News/sent-mail/")))
```

Gnus will insert `Gcc` headers in all outgoing messages that point to one or more group(s) on that server. Which group to use is determined by the `gnus-message-archive-group` variable.

This variable can be:

- a string Messages will be saved in that group.
- a list of strings Messages will be saved in all those groups.
- an alist of regexps, functions and forms When a key "matches", the result is used.

Let's illustrate:

Just saving to a single group called 'MisK':

```
(setq gnus-message-archive-group "MisK")
```

Saving to two groups, 'MisK' and 'safe':

```
(setq gnus-message-archive-group '("MisK" "safe"))
```

Save to different groups based on what group you are in:

```
(setq gnus-message-archive-group
      '(("^alt" "sent-to-alt")
        ("mail" "sent-to-mail")
        (".*" "sent-to-misc")))
```

More complex stuff:

```
(setq gnus-message-archive-group
      '((if (message-news-p)
            "misc-news"
          "misc-mail")))
```

This is the default.

How about storing all news messages in one file, but storing all mail messages in one file per month:

```
(setq gnus-message-archive-group
      '((if (message-news-p)
            "misc-news"
          (concat "mail." (format-time-string
                           "%Y-%m" (current-time))))))
```

Now, when you send a message off, it will be stored in the appropriate group. (If you want to disable storing for just one particular message, you can just remove the `Gcc` header that has been inserted.) The archive group will appear in the group buffer the next time you start Gnus, or the next time you press *F* in the group buffer. You can enter it and read the articles in it just like you'd read any other group. If the group gets really big and annoying, you can simply rename if (using *G r* in the group buffer) to something nice – '`misc-mail-september-1995`', or whatever. New messages will continue to be stored in the old (now empty) group.

That's the default method of archiving sent mail. Gnus also offers two other variables for the people who don't like the default method. In that case you should set `gnus-message-archive-group` to `nil`; this will disable archiving.

XEmacs 19.13 doesn't have `format-time-string`, so you'll have to use a different value for `gnus-message-archive-group` there.

`gnus-outgoing-message-group`

All outgoing messages will be put in this group. If you want to store all your outgoing mail and articles in the group '`nnml:archive`', you set this variable to that value. This variable can also be a list of group names.

If you want to have greater control over what group to put each message in, you can set this variable to a function that checks the current newsgroup name and then returns a suitable group name (or list of names).

6 Select Methods

A *foreign group* is a group that is not read by the usual (or default) means. It could be, for instance, a group from a different NNTP server, it could be a virtual group, or it could be your own personal mail group.

A foreign group (or any group, really) is specified by a *name* and a *select method*. To take the latter first, a select method is a list where the first element says what backend to use (eg. `nntp`, `nnspool`, `nnml`) and the second element is the *server name*. There may be additional elements in the select method, where the value may have special meaning for the backend in question.

One could say that a select method defines a *virtual server*—so we do just that (see Section 6.1 [The Server Buffer], page 77).

The *name* of the group is the name the backend will recognize the group as.

For instance, the group '`soc.motss`' on the NNTP server '`some.where.edu`' will have the name '`soc.motss`' and select method `(nntp "some.where.edu")`. Gnus will call this group, in all circumstances, '`nntp+some.where.edu:soc.motss`', even though the `nntp` backend just knows this group as '`soc.motss`'.

The different methods all have their peculiarities, of course.

6.1 The Server Buffer

Traditionally, a *server* is a machine or a piece of software that one connects to, and then requests information from. Gnus does not connect directly to any real servers, but does all transactions through one backend or other. But that's just putting one layer more between the actual media and Gnus, so we might just as well say that each backend represents a virtual server.

For instance, the `nntp` backend may be used to connect to several different actual NNTP servers, or, perhaps, to many different ports on the same actual NNTP server. You tell Gnus which backend to use, and what parameters to set by specifying a *select method*.

These select methods specifications can sometimes become quite complicated—say, for instance, that you want to read from the NNTP server '`news.funet.fi`' on port number `13`, which hangs if queried for NOV headers and has a buggy select. Ahem. Anyways, if you had to specify that for each group that used this server, that would be too much work, so Gnus offers a way of naming select methods, which is what you do in the server buffer.

To enter the server buffer, user the `^` (`gnus-group-enter-server-mode`) command in the group buffer.

`gnus-server-mode-hook` is run when creating the server buffer.

6.1.1 Server Buffer Format

You can change the look of the server buffer lines by changing the `gnus-server-line-format` variable. This is a `format`-like variable, with some simple extensions:

‘h’ How the news is fetched—the backend name.

‘n’ The name of this server.

‘w’ Where the news is to be fetched from—the address.

‘s’ The opened/closed/denied status of the server.

The mode line can also be customized by using the `gnus-server-mode-line-format` variable. The following specs are understood:

‘S’ Server name.

‘M’ Server method.

Also see Section 8.3 [Formatting Variables], page 120.

6.1.2 Server Commands

a	Add a new server (`gnus-server-add-server`).
e	Edit a server (`gnus-server-edit-server`).
SPACE	Browse the current server (`gnus-server-read-server`).
q	Return to the group buffer (`gnus-server-exit`).
k	Kill the current server (`gnus-server-kill-server`).
y	Yank the previously killed server (`gnus-server-yank-server`).
c	Copy the current server (`gnus-server-copy-server`).
l	List all servers (`gnus-server-list-servers`).

6.1.3 Example Methods

Most select methods are pretty simple and self-explanatory:

```
(nntp "news.funet.fi")
```

Reading directly from the spool is even simpler:

```
(nnspool "")
```

As you can see, the first element in a select method is the name of the backend, and the second is the *address*, or *name*, if you will.

After these two elements, there may be a arbitrary number of *(variable form)* pairs.

To go back to the first example—imagine that you want to read from port `15` from that machine. This is what the select method should look like then:

```
(nntp "news.funet.fi" (nntp-port-number 15))
```

You should read the documentation to each backend to find out what variables are relevant, but here's an `nnmh` example.

`nnmh` is a mail backend that reads a spool-like structure. Say you have two structures that you wish to access: One is your private mail spool, and the other is a public one. Here's the possible spec for you private mail:

```
(nnmh "private" (nnmh-directory "~/private/mail/"))
```

(This server is then called '`private`', but you may have guessed that.)

Here's the method for a public spool:

```
(nnmh "public"
      (nnmh-directory "/usr/information/spool/")
      (nnmh-get-new-mail nil))
```

6.1.4 Creating a Virtual Server

If you're saving lots of articles in the cache by using persistent articles, you may want to create a virtual server to read the cache.

First you need to add a new server. The `a` command does that. It would probably be best to use `nnspool` to read the cache. You could also use `nnml` or `nnmh`, though.

Type *`a nnspool RET cache RET`*.

You should now have a brand new `nnspool` virtual server called '`cache`'. You now need to edit it to have the right definitions. Type `e` to edit the server. You'll be entered into a buffer that will contain the following:

```
(nnspool "cache")
```

Change that to:

```
(nnspool "cache"
 (nnspool-spool-directory "~/News/cache/")
 (nnspool-nov-directory "~/News/cache/")
 (nnspool-active-file "~/News/cache/active"))
```

Type *`C-c C-c`* to return to the server buffer. If you now press *`RET`* over this virtual server, you should be entered into a browse buffer, and you should be able to enter any of the groups displayed.

6.1.5 Servers and Methods

Wherever you would normally use a select method (eg. `gnus-secondary-select-method`, in the group select method, when browsing a foreign server) you can use a virtual server name instead. This could potentially save lots of typing. And it's nice all over.

6.1.6 Unavailable Servers

If a server seems to be unreachable, Gnus will mark that server as `denied`. That means that any subsequent attempt to make contact with that server will just be ignored. "It can't be opened," Gnus will tell you, without making the least effort to see whether that is actually the case or not.

That might seem quite naughty, but it does make sense most of the time. Let's say you have 10 groups subscribed to the server '`nepholococcygia.com`'. This server is located somewhere quite far away from you, the machine is quite, so it takes 1 minute just to find

out that it refuses connection from you today. If Gnus were to attempt to do that 10 times, you'd be quite annoyed, so Gnus won't attempt to do that. Once it has gotten a single "connection refused", it will regard that server as "down".

So, what happens if the machine was only feeling unwell temporarily? How do you test to see whether the machine has come up again?

You jump to the server buffer (see Section 6.1 [The Server Buffer], page 77) and poke it with the following commands:

`O` Try to establish connection to the server on the current line (`gnus-server-open-server`).

`C` Close the connection (if any) to the server (`gnus-server-close-server`).

`D` Mark the current server as unreachable (`gnus-server-deny-server`).

`R` Remove all marks to whether Gnus was denied connection from all servers (`gnus-server-remove-denials`).

6.2 Getting News

A newsreader is normally used for reading news. Gnus currently provides only two methods of getting news – it can read from an NNTP server, or it can read from a local spool.

6.2.1 NNTP

Subscribing to a foreign group from an NNTP server is rather easy. You just specify `nntp` as method and the address of the NNTP server as the, uhm, address.

If the NNTP server is located at a non-standard port, setting the third element of the select method to this port number should allow you to connect to the right port. You'll have to edit the group info for that (see Section 2.8 [Foreign Groups], page 18).

The name of the foreign group can be the same as a native group. In fact, you can subscribe to the same group from as many different servers you feel like. There will be no name collisions.

The following variables can be used to create a virtual `nntp` server:

`nntp-server-opened-hook`
: `nntp-server-opened-hook` is run after a connection has been made. It can be used to send commands to the NNTP server after it has been contacted. By default is sends the command `MODE READER` to the server with the `nntp-send-mode-reader` function. Another popular function is `nntp-send-authinfo`, which will prompt you for an NNTP password and stuff.

`nntp-server-action-alist`
: This is an list of regexps to match on server types and actions to be taken when matches are made. For instance, if you want Gnus to beep every time you connect to innd, you could say something like:

```
(setq nntp-server-action-alist
      '(("innd" (ding))))
```

You probably don't want to do that, though.

The default value is

```
'(("nntpd 1\\.5\\.11t"
   (remove-hook 'nntp-server-opened-hook nntp-send-mode-reader)))
```

This ensures that Gnus doesn't send the `MODE READER` command to nntpd 1.5.11t, since that command chokes that server, I've been told.

`nntp-maximum-request`

If the NNTP server doesn't support NOV headers, this backend will collect headers by sending a series of `head` commands. To speed things up, the backend sends lots of these commands without waiting for reply, and then reads all the replies. This is controlled by the `nntp-maximum-request` variable, and is 400 by default. If your network is buggy, you should set this to 1.

`nntp-connection-timeout`

If you have lots of foreign `nntp` groups that you connect to regularly, you're sure to have problems with NNTP servers not responding properly, or being too loaded to reply within reasonable time. This is can lead to awkward problems, which can be helped somewhat by setting `nntp-connection-timeout`. This is an integer that says how many seconds the `nntp` backend should wait for a connection before giving up. If it is `nil`, which is the default, no timeouts are done.

`nntp-command-timeout`

If you're running Gnus on a machine that has a dynamically assigned address, Gnus may become confused. If the address of your machine changes after connecting to the NNTP server, Gnus will simply sit waiting forever for replies from the server. To help with this unfortunate problem, you can set this command to a number. Gnus will then, if it sits waiting longer than that number of seconds for a reply from the server, shut down the connection, start a new one, and resend the command. This should hopefully be transparent to the user. A likely number is 30 seconds.

`nntp-retry-on-break`

If this variable is non-`nil`, you can also *C-g* if Gnus hangs. This will have much the same effect as the command timeout described above.

`nntp-server-hook`

This hook is run as the last step when connecting to an NNTP server.

`nntp-open-server-function`

This function is used to connect to the remote system. Two pre-made functions are `nntp-open-network-stream`, which is the default, and simply connects to some port or other on the remote system. The other is `nntp-open-rlogin`, which does an rlogin on the remote system, and then does a telnet to the NNTP server available there.

`nntp-rlogin-parameters`
: If you use `nntp-open-rlogin` as the `nntp-open-server-function`, this list will be used as the parameter list given to `rsh`.

`nntp-end-of-line`
: String to use as end-of-line markers when talking to the NNTP server. This is '`\r\n`' by default, but should be '`\n`' when using `rlogin` to talk to the server.

`nntp-rlogin-user-name`
: User name on the remote system when using the `rlogin` connect function.

`nntp-address`
: The address of the remote system running the NNTP server.

`nntp-port-number`
: Port number to connect to when using the `nntp-open-network-stream` connect function.

`nntp-buggy-select`
: Set this to non-`nil` if your select routine is buggy.

`nntp-nov-is-evil`
: If the NNTP server does not support NOV, you could set this variable to `t`, but `nntp` usually checks whether NOV can be used automatically.

`nntp-xover-commands`
: List of strings that are used as commands to fetch NOV lines from a server. The default value of this variable is `("XOVER" "XOVERVIEW")`.

`nntp-nov-gap`
: `nntp` normally sends just one big request for NOV lines to the server. The server responds with one huge list of lines. However, if you have read articles 2-5000 in the group, and only want to read article 1 and 5001, that means that `nntp` will fetch 4999 NOV lines that you do not want, and will not use. This variable says how big a gap between two consecutive articles is allowed to be before the `XOVER` request is split into several request. Note that if your network is fast, setting this variable to a really small number means that fetching will probably be slower. If this variable is `nil`, `nntp` will never split requests.

`nntp-prepare-server-hook`
: A hook run before attempting to connect to an NNTP server.

`nntp-async-number`
: How many articles should be pre-fetched when in asynchronous mode. If this variable is `t`, `nntp` will pre-fetch all the articles that it can without bound. If it is `nil`, no pre-fetching will be made.

`nntp-warn-about-losing-connection`
: If this variable is non-`nil`, some noise will be made when a server closes connection.

6.2.2 News Spool

Subscribing to a foreign group from the local spool is extremely easy, and might be useful, for instance, to speed up reading groups like ‘`alt.binaries.pictures.furniture`’.

Anyways, you just specify `nnspool` as the method and ‘’ (or anything else) as the address.

If you have access to a local spool, you should probably use that as the native select method (see Section 1.1 [Finding the News], page 3). It is normally faster than using an `nntp` select method, but might not be. It depends. You just have to try to find out what's best at your site.

`nnspool-inews-program`
: Program used to post an article.

`nnspool-inews-switches`
: Parameters given to the inews program when posting an article.

`nnspool-spool-directory`
: Where `nnspool` looks for the articles. This is normally ‘`/usr/spool/news/`’.

`nnspool-nov-directory`
: Where `nnspool` will look for NOV files. This is normally ‘`/usr/spool/news/over.view/`’.

`nnspool-lib-dir`
: Where the news lib dir is (‘`/usr/lib/news/`’ by default).

`nnspool-active-file`
: The path of the active file.

`nnspool-newsgroups-file`
: The path of the group descriptions file.

`nnspool-history-file`
: The path of the news history file.

`nnspool-active-times-file`
: The path of the active date file.

`nnspool-nov-is-evil`
: If non-`nil`, `nnspool` won't try to use any NOV files that it finds.

`nnspool-sift-nov-with-sed`
: If non-`nil`, which is the default, use `sed` to get the relevant portion from the overview file. If nil, `nnspool` will load the entire file into a buffer and process it there.

6.3 Getting Mail

Reading mail with a newsreader—isn't that just plain WeIrD? But of course.

6.3.1 Getting Started Reading Mail

It's quite easy to use Gnus to read your new mail. You just plonk the mail backend of your choice into `gnus-secondary-select-methods`, and things will happen automatically.

For instance, if you want to use `nnml` (which is a one file per mail backend), you could put the following in your '`.gnus`' file:

```
(setq gnus-secondary-select-methods
      '((nnml "private")))
```

Now, the next time you start Gnus, this backend will be queried for new articles, and it will move all the messages in your spool file to its directory, which is `~/Mail/` by default. The new group that will be created ('`mail.misc`') will be subscribed, and you can read it like any other group.

You will probably want to split the mail into several groups, though:

```
(setq nnmail-split-methods
  '(("junk" "^From:.*Lars Ingebrigtsen")
    ("crazy" "^Subject:.*die\\|^Organization:.*flabby")
    ("other" "")))
```

This will result in three new mail groups being created: '`nnml:junk`', '`nnml:crazy`', and '`nnml:other`'. All the mail that doesn't fit into the first two groups will be placed in the latter group.

This should be sufficient for reading mail with Gnus. You might want to give the other sections in this part of the manual a perusal, though, especially see Section 6.3.10 [Choosing a Mail Backend], page 92 and see Section 6.3.7 [Expiring Mail], page 89.

6.3.2 Splitting Mail

The `nnmail-split-methods` variable says how the incoming mail is to be split into groups.

```
(setq nnmail-split-methods
  '(("mail.junk" "^From:.*Lars Ingebrigtsen")
    ("mail.crazy" "^Subject:.*die\\|^Organization:.*flabby")
    ("mail.other" "")))
```

This variable is a list of lists, where the first element of each of these lists is the name of the mail group (they do not have to be called something beginning with '`mail`', by the way), and the second element is a regular expression used on the header of each mail to determine if it belongs in this mail group.

The second element can also be a function. In that case, it will be called narrowed to the headers with the first element of the rule as the argument. It should return a non-`nil` value if it thinks that the mail belongs in that group.

The last of these groups should always be a general one, and the regular expression should *always* be '' so that it matches any mails that haven't been matched by any of the other regexps.

If you like to tinker with this yourself, you can set this variable to a function of your choice. This function will be called without any arguments in a buffer narrowed to the headers of an incoming mail message. The function should return a list of groups names that it thinks should carry this mail message.

Note that the mail backends are free to maul the poor, innocent incoming headers all they want to. They all add `Lines` headers; some add `X-Gnus-Group` headers; most rename the Unix mbox `From<SPACE>` line to something else.

The mail backends all support cross-posting. If several regexps match, the mail will be "cross-posted" to all those groups. `nnmail-crosspost` says whether to use this mechanism or not. Note that no articles are crossposted to the general (‘’) group.

`nnmh` and `nnml` makes crossposts by creating hard links to the crossposted articles. However, not all files systems support hard links. If that's the case for you, set `nnmail-crosspost-link-function` to `copy-file`. (This variable is `add-name-to-file` by default.)

Gnus gives you all the opportunity you could possibly want for shooting yourself in the foot. Let's say you create a group that will contain all the mail you get from your boss. And then you accidentally unsubscribe from the group. Gnus will still put all the mail from your boss in the unsubscribed group, and so, when your boss mails you "Have that report ready by Monday or you're fired!", you'll never see it and, come Tuesday, you'll still believe that you're gainfully employed while you really should be out collecting empty bottles to save up for next month's rent money.

6.3.3 Mail Backend Variables

These variables are (for the most part) pertinent to all the various mail backends.

`nnmail-read-incoming-hook`

The mail backends all call this hook after reading new mail. You can use this hook to notify any mail watch programs, if you want to.

`nnmail-spool-file`

The backends will look for new mail in this file. If this variable is `nil`, the mail backends will never attempt to fetch mail by themselves. If you are using a POP mail server and your name is ‘`larsi`’, you should set this variable to ‘`po:larsi`’. If your name is not ‘`larsi`’, you should probably modify that slightly, but you may have guessed that already, you smart & handsome devil! You can also set this variable to `pop`, and Gnus will try to figure out the POP mail string by itself. In any case, Gnus will call `movemail` which will contact the POP server named in the `MAILHOST` environment variable. If the POP server needs a password, you can either set `nnmail-pop-password-required` to `t` and be prompted for the password, or set `nnmail-pop-password` to the password itself.

When you use a mail backend, Gnus will slurp all your mail from your inbox and plonk it down in your home directory. Gnus doesn't move any mail if you're not using a mail backend—you have to do a lot of magic invocations first. At the time when you have finished drawing the pentagram, lightened the candles,

and sacrificed the goat, you really shouldn't be too surprised when Gnus moves your mail.

`nnmail-use-procmail`
: If non-`nil`, the mail backends will look in `nnmail-procmail-directory` for incoming mail. All the files in that directory that have names ending in `nnmail-procmail-suffix` will be considered incoming mailboxes, and will be searched for new mail.

`nnmail-crash-box`
: When the mail backends read a spool file, it is first moved to this file, which is '`~/.gnus-crash-box`' by default. If this file already exists, it will always be read (and incorporated) before any other spool files.

`nnmail-prepare-incoming-hook`
: This is run in a buffer that holds all the new incoming mail, and can be used for, well, anything, really.

`nnmail-pre-get-new-mail-hook`
`nnmail-post-get-new-mail-hook`
: These are two useful hooks executed when treating new incoming mail—`nnmail-pre-get-new-mail-hook` (is called just before starting to handle the new mail) and `nnmail-post-get-new-mail-hook` (is called when the mail handling is done). Here's and example of using these two hooks to change the default file modes the new mail files get:

```
(add-hook 'gnus-pre-get-new-mail-hook
          (lambda () (set-default-file-modes 511)))

(add-hook 'gnus-post-get-new-mail-hook
          (lambda () (set-default-file-modes 551)))
```

`nnmail-tmp-directory`
: This variable says where to move the incoming mail to while processing it. This is usually done in the same directory that the mail backend inhabits (i.e., '`~/Mail/`'), but if this variable is non-`nil`, it will be used instead.

`nnmail-movemail-program`
: This program is executed to move mail from the user's inbox to her home directory. The default is '`movemail`'.

`nnmail-delete-incoming`
: If non-`nil`, the mail backends will delete the temporary incoming file after splitting mail into the proper groups. This is `nil` by default for reasons of security.

`nnmail-use-long-file-names`
: If non-`nil`, the mail backends will use long file and directory names. Groups like '`mail.misc`' will end up in directories like '`mail.misc/`'. If it is `nil`, the same group will end up in '`mail/misc/`'.

`nnmail-delete-file-function`
: Function called to delete files. It is `delete-file` by default.

6.3.4 Fancy Mail Splitting

If the rather simple, standard method for specifying how to split mail doesn't allow you to do what you want, you can set `nnmail-split-methods` to `nnmail-split-fancy`. Then you can play with the `nnmail-split-fancy` variable.

Let's look at an example value of this variable first:

```
;; Messages from the mailer daemon are not crossposted to any of
;; the ordinary groups.  Warnings are put in a separate group
;; from real errors.
(| ("from" mail (| ("subject" "warn.*" "mail.warning")
                   "mail.misc"))
   ;; Non-error messages are crossposted to all relevant
   ;; groups, but we don't crosspost between the group for the
   ;; (ding) list and the group for other (ding) related mail.
   (& (| (any "ding@ifi\\.uio\\.no" "ding.list")
         ("subject" "ding" "ding.misc"))
      ;; Other mailing lists...
      (any "procmail@informatik\\.rwth-aachen\\.de" "procmail.list")
      (any "SmartList@informatik\\.rwth-aachen\\.de" "SmartList.list")
      ;; People...
      (any "larsi@ifi\\.uio\\.no" "people.Lars Magne Ingebrigtsen"))
   ;; Unmatched mail goes to the catch all group.
   "misc.misc"))")
```

This variable has the format of a *split*. A split is a (possibly) recursive structure where each split may contain other splits. Here are the four possible split syntaxes:

GROUP If the split is a string, that will be taken as a group name.

(FIELD VALUE SPLIT)
: If the split is a list, and the first element is a string, then that means that if header FIELD (a regexp) contains VALUE (also a regexp), then store the message as specified by SPLIT.

(| SPLIT...)
: If the split is a list, and the first element is `|` (vertical bar), then process each SPLIT until one of them matches. A SPLIT is said to match if it will cause the mail message to be stored in one or more groups.

(& SPLIT...)
: If the split is a list, and the first element is `&`, then process all SPLITs in the list.

In these splits, FIELD must match a complete field name. VALUE must match a complete word according to the fundamental mode syntax table. You can use `.*` in the regexps to match partial field names or words.

FIELD and VALUE can also be lisp symbols, in that case they are expanded as specified by the variable `nnmail-split-abbrev-alist`. This is an alist of cons cells, where the car of the cells contains the key, and the cdr contains a string.

`nnmail-split-fancy-syntax-table` is the syntax table in effect when all this splitting is performed.

6.3.5 Mail and Procmail

Many people use `procmail` (or some other mail filter program or external delivery agent—`slocal`, `elm`, etc) to split incoming mail into groups. If you do that, you should set `nnmail-spool-file` to `procmail` to ensure that the mail backends never ever try to fetch mail by themselves.

This also means that you probably don't want to set `nnmail-split-methods` either, which has some, perhaps, unexpected side effects.

When a mail backend is queried for what groups it carries, it replies with the contents of that variable, along with any groups it has figured out that it carries by other means. None of the backends (except `nnmh`) actually go out to the disk and check what groups actually exist. (It's not trivial to distinguish between what the user thinks is a basis for a newsgroup and what is just a plain old file or directory.)

This means that you have to tell Gnus (and the backends) what groups exist by hand.

Let's take the `nnmh` backend as an example.

The folders are located in `nnmh-directory`, say, '`~/Mail/`'. There are three folders, '`foo`', '`bar`' and '`mail.baz`'.

Go to the group buffer and type *G m*. When prompted, answer '`foo`' for the name and '`nnmh`' for the method. Repeat twice for the two other groups, '`bar`' and '`mail.baz`'. Be sure to include all your mail groups.

That's it. You are now set to read your mail. An active file for this method will be created automatically.

If you use `nnfolder` or any other backend that store more than a single article in each file, you should never have procmail add mails to the file that Gnus sees. Instead, procmail should put all incoming mail in `nnmail-procmail-directory`. To arrive at the file name to put the incoming mail in, append `nnmail-procmail-suffix` to the group name. The mail backends will read the mail from these files.

When Gnus reads a file called '`mail.misc.spool`', this mail will be put in the `mail.misc`, as one would expect. However, if you want Gnus to split the mail the normal way, you could set `nnmail-resplit-incoming` to `t`.

If you use `procmail` to split things directory into an `nnmh` directory (which you shouldn't do), you should set `nnmail-keep-last-article` to non-`nil` to prevent Gnus from ever expiring the final article in a mail newsgroup. This is quite, quite important.

6.3.6 Incorporating Old Mail

Most people have lots of old mail stored in various file formats. If you have set up Gnus to read mail using one of the spiffy Gnus mail backends, you'll probably wish to have that old mail incorporated into your mail groups.

Doing so can be quite easy.

To take an example: You're reading mail using `nnml` (see Section 6.3.10.3 [Mail Spool], page 92), and have set `nnmail-split-methods` to a satisfactory value (see Section 6.3.2 [Splitting Mail], page 84). You have an old Unix mbox file filled with important, but old, mail. You want to move it into your `nnml` groups.

Here's how:

1. Go to the group buffer.
2. Type 'G f' and give the path of the mbox file when prompted to create an `nndoc` group from the mbox file (see Section 2.8 [Foreign Groups], page 18).
3. Type 'SPACE' to enter the newly created group.
4. Type 'M P b' to process-mark all articles in this group (see Section 3.7.5 [Setting Process Marks], page 39).
5. Type 'B r' to respool all the process-marked articles, and answer '`nnml`' when prompted (see Section 3.22 [Mail Group Commands], page 63).

All the mail messages in the mbox file will now also be spread out over all your `nnml` groups. Try entering them and check whether things have gone without a glitch. If things look ok, you may consider deleting the mbox file, but I wouldn't do that unless I was absolutely sure that all the mail has ended up where it should be.

Respooling is also a handy thing to do if you're switching from one mail backend to another. Just respool all the mail in the old mail groups using the new mail backend.

6.3.7 Expiring Mail

Traditional mail readers have a tendency to remove mail articles when you mark them as read, in some way. Gnus takes a fundamentally different approach to mail reading.

Gnus basically considers mail just to be news that has been received in a rather peculiar manner. It does not think that it has the power to actually change the mail, or delete any mail messages. If you enter a mail group, and mark articles as "read", or kill them in some other fashion, the mail articles will still exist on the system. I repeat: Gnus will not delete your old, read mail. Unless you ask it to, of course.

To make Gnus get rid of your unwanted mail, you have to mark the articles as *expirable*. This does not mean that the articles will disappear right away, however. In general, a mail article will be deleted from your system if, 1) it is marked as expirable, AND 2) it is more than one week old. If you do not mark an article as expirable, it will remain on your system until hell freezes over. This bears repeating one more time, with some spurious capitalizations: IF you do NOT mark articles as EXPIRABLE, Gnus will NEVER delete those ARTICLES.

You do not have to mark articles as expirable by hand. Groups that match the regular expression `gnus-auto-expirable-newsgroups` will have all articles that you read marked as expirable automatically. All articles that are marked as expirable have an '`E`' in the first column in the summary buffer.

Let's say you subscribe to a couple of mailing lists, and you want the articles you have read to disappear after a while:

```
(setq gnus-auto-expirable-newsgroups
      "mail.nonsense-list\\|mail.nice-list")
```

Another way to have auto-expiry happen is to have the element `auto-expire` in the group parameters of the group.

The `nnmail-expiry-wait` variable supplies the default time an expirable article has to live. The default is seven days.

Gnus also supplies a function that lets you fine-tune how long articles are to live, based on what group they are in. Let's say you want to have one month expiry period in the '`mail.private`' group, a one day expiry period in the '`mail.junk`' group, and a six day expiry period everywhere else:

```
(setq nnmail-expiry-wait-function
      (lambda (group)
       (cond ((string= group "mail.private")
              31)
             ((string= group "mail.junk")
              1)
             ((string= group "important")
              'never)
             (t
              6))))
```

The group names that this function is fed are "unadorned" group names—no '`nnml:`' prefixes and the like.

The `nnmail-expiry-wait` variable and `nnmail-expiry-wait-function` function can be either a number (not necessarily an integer) or the symbols `immediate` or `never`.

You can also use the `expiry-wait` group parameter to selectively change the expiry period (see Section 2.9 [Group Parameters], page 19).

If `nnmail-keep-last-article` is non-`nil`, Gnus will never expire the final article in a mail newsgroup. This is to make life easier for procmail users.

By the way, that line up there about Gnus never expiring non-expirable articles is a lie. If you put `total-expire` in the group parameters, articles will not be marked as expirable, but all read articles will be put through the expiry process. Use with extreme caution. Even more dangerous is the `gnus-total-expirable-newsgroups` variable. All groups that match this regexp will have all read articles put through the expiry process, which means that *all* old mail articles in the groups in question will be deleted after a while. Use with extreme caution, and don't come crying to me when you discover that the regexp you used matched the wrong group and all your important mail has disappeared. Be a *man*! Or a *woman*! Whatever you feel more comfortable with! So there!

6.3.8 Duplicates

If you are a member of a couple of mailing list, you will sometime receive two copies of the same mail. This can be quite annoying, so `nnmail` checks for and treats any duplicates it might find. To do this, it keeps a cache of old `Message-ID`s - `nnmail-message-id-cache-file`, which is '`~/.nnmail-cache`' by default. The approximate maximum number of

`Message-ID`s stored there is controlled by the `nnmail-message-id-cache-length` variable, which is 1000 by default. (So 1000 `Message-ID`s will be stored.) If all this sounds scary to you, you can set `nnmail-treat-duplicates` to `warn` (which is what it is by default), and `nnmail` won't delete duplicate mails. Instead it will generate a brand new `Message-ID` for the mail and insert a warning into the head of the mail saying that it thinks that this is a duplicate of a different message.

This variable can also be a function. If that's the case, the function will be called from a buffer narrowed to the message in question with the `Message-ID` as a parameter. The function must return either `nil`, `warn`, or `delete`.

You can turn this feature off completely by setting the variable to `nil`.

If you want all the duplicate mails to be put into a special *duplicates* group, you could do that using the normal mail split methods:

```
(setq nnmail-split-fancy
      '(| ;; Messages duplicates go to a separate group.
          ("gnus-warning" "duplication of message" "duplicate")
          ;; Message from daemons, postmaster, and the like to another.
          (any mail "mail.misc")
          ;; Other rules.
          [ ... ] ))
```

Or something like:

```
(setq nnmail-split-methods
      '(("duplicates" "^Gnus-Warning:")
        ;; Other rules.
        [...]))
```

Here's a neat feature: If you know that the recipient reads her mail with Gnus, and that she has `nnmail-treat-duplicates` set to `delete`, you can send her as many insults as you like, just by using a `Message-ID` of a mail that you know that she's already received. Think of all the fun! She'll never see any of it! Whee!

6.3.9 Not Reading Mail

If you start using any of the mail backends, they have the annoying habit of assuming that you want to read mail with them. This might not be unreasonable, but it might not be what you want.

If you set `nnmail-spool-file` to `nil`, none of the backends will ever attempt to read incoming mail, which should help.

This might be too much, if, for instance, you are reading mail quite happily with `nnml` and just want to peek at some old RMAIL file you have stashed away with `nnbabyl`. All backends have variables called backend-`get-new-mail`. If you want to disable the `nnbabyl` mail reading, you edit the virtual server for the group to have a setting where `nnbabyl-get-new-mail` to `nil`.

All the mail backends will call `nn*-prepare-save-mail-hook` narrowed to the article to be saved before saving it when reading incoming mail.

6.3.10 Choosing a Mail Backend

Gnus will read the mail spool when you activate a mail group. The mail file is first copied to your home directory. What happens after that depends on what format you want to store your mail in.

6.3.10.1 Unix Mail Box

The *nnmbox* backend will use the standard Un*x mbox file to store mail. `nnmbox` will add extra headers to each mail article to say which group it belongs in.

Virtual server settings:

`nnmbox-mbox-file`
: The name of the mail box in the user's home directory.

`nnmbox-active-file`
: The name of the active file for the mail box.

`nnmbox-get-new-mail`
: If non-`nil`, `nnmbox` will read incoming mail and split it into groups.

6.3.10.2 Rmail Babyl

The *nnbabyl* backend will use a babyl mail box (aka. *rmail mbox*) to store mail. `nnbabyl` will add extra headers to each mail article to say which group it belongs in.

Virtual server settings:

`nnbabyl-mbox-file`
: The name of the rmail mbox file.

`nnbabyl-active-file`
: The name of the active file for the rmail box.

`nnbabyl-get-new-mail`
: If non-`nil`, `nnbabyl` will read incoming mail.

6.3.10.3 Mail Spool

The *nnml* spool mail format isn't compatible with any other known format. It should be used with some caution.

If you use this backend, Gnus will split all incoming mail into files; one file for each mail, and put the articles into the correct directories under the directory specified by the `nnml-directory` variable. The default value is '`~/Mail/`'.

You do not have to create any directories beforehand; Gnus will take care of all that.

If you have a strict limit as to how many files you are allowed to store in your account, you should not use this backend. As each mail gets its own file, you might very well occupy

thousands of inodes within a few weeks. If this is no problem for you, and it isn't a problem for you having your friendly systems administrator walking around, madly, shouting "Who is eating all my inodes?! Who? Who!?!", then you should know that this is probably the fastest format to use. You do not have to trudge through a big mbox file just to read your new mail.

`nnml` is probably the slowest backend when it comes to article splitting. It has to create lots of files, and it also generates NOV databases for the incoming mails. This makes is the fastest backend when it comes to reading mail.

Virtual server settings:

`nnml-directory`
: All `nnml` directories will be placed under this directory.

`nnml-active-file`
: The active file for the `nnml` server.

`nnml-newsgroups-file`
: The `nnml` group descriptions file. See Section 10.5.7.2 [Newsgroups File Format], page 158.

`nnml-get-new-mail`
: If non-`nil`, `nnml` will read incoming mail.

`nnml-nov-is-evil`
: If non-`nil`, this backend will ignore any NOV files.

`nnml-nov-file-name`
: The name of the NOV files. The default is '`.overview`'.

`nnml-prepare-save-mail-hook`
: Hook run narrowed to an article before saving.

If your `nnml` groups and NOV files get totally out of whack, you can do a complete update by typing *`M-x nnml-generate-nov-databases`*. This command will trawl through the entire `nnml` hierarchy, looking at each and every article, so it might take a while to complete.

6.3.10.4 MH Spool

`nnmh` is just like `nnml`, except that is doesn't generate NOV databases and it doesn't keep an active file. This makes `nnmh` a *much* slower backend than `nnml`, but it also makes it easier to write procmail scripts for.

Virtual server settings:

`nnmh-directory`
: All `nnmh` directories will be located under this directory.

`nnmh-get-new-mail`
: If non-`nil`, `nnmh` will read incoming mail.

`nnmh-be-safe`
: If non-`nil`, `nnmh` will go to ridiculous lengths to make sure that the articles in the folder are actually what Gnus thinks they are. It will check date stamps and stat everything in sight, so setting this to `t` will mean a serious slow-down. If you never use anything but Gnus to read the `nnmh` articles, you do not have to set this variable to `t`.

6.3.10.5 Mail Folders

`nnfolder` is a backend for storing each mail group in a separate file. Each file is in the standard Un*x mbox format. `nnfolder` will add extra headers to keep track of article numbers and arrival dates.

Virtual server settings:

`nnfolder-directory`
: All the `nnfolder` mail boxes will be stored under this directory.

`nnfolder-active-file`
: The name of the active file.

`nnfolder-newsgroups-file`
: The name of the group descriptions file. See Section 10.5.7.2 [Newsgroups File Format], page 158.

`nnfolder-get-new-mail`
: If non-`nil`, `nnfolder` will read incoming mail.

If you have lots of `nnfolder`-like files you'd like to read with `nnfolder`, you can use the *`M-x nnfolder-generate-active-file`* command to make `nnfolder` aware of all likely files in `nnfolder-directory`.

6.4 Other Sources

Gnus can do more than just read news or mail. The methods described below allow Gnus to view directories and files as if they were newsgroups.

6.4.1 Directory Groups

If you have a directory that has lots of articles in separate files in it, you might treat it as a newsgroup. The files have to have numerical names, of course.

This might be an opportune moment to mention `ange-ftp`, that most wonderful of all wonderful Emacs packages. When I wrote `nndir`, I didn't think much about it—a backend to read directories. Big deal.

`ange-ftp` changes that picture dramatically. For instance, if you enter '`"/ftp.hpc.uh.edu:/pub/emacs/dir`
as the the directory name, ange-ftp will actually allow you to read this directory over at '`sina`' as a newsgroup. Distributed news ahoy!

`nndir` will use NOV files if they are present.

`nndir` is a "read-only" backend—you can't delete or expire articles with this method. You can use `nnmh` or `nnml` for whatever you use `nndir` for, so you could switch to any of those methods if you feel the need to have a non-read-only `nndir`.

6.4.2 Anything Groups

From the `nndir` backend (which reads a single spool-like directory), it's just a hop and a skip to `nneething`, which pretends that any arbitrary directory is a newsgroup. Strange, but true.

When `nneething` is presented with a directory, it will scan this directory and assign article numbers to each file. When you enter such a group, `nneething` must create "headers" that Gnus can use. After all, Gnus is a newsreader, in case you're forgetting. `nneething` does this in a two-step process. First, it snoops each file in question. If the file looks like an article (i.e., the first few lines look like headers), it will use this as the head. If this is just some arbitrary file without a head (eg. a C source file), `nneething` will cobble up a header out of thin air. It will use file ownership, name and date and do whatever it can with these elements.

All this should happen automatically for you, and you will be presented with something that looks very much like a newsgroup. Totally like a newsgroup, to be precise. If you select an article, it will be displayed in the article buffer, just as usual.

If you select a line that represents a directory, Gnus will pop you into a new summary buffer for this `nneething` group. And so on. You can traverse the entire disk this way, if you feel like, but remember that Gnus is not dired, really, and does not intend to be, either.

There are two overall modes to this action—ephemeral or solid. When doing the ephemeral thing (i.e., *G D* from the group buffer), Gnus will not store information on what files you have read, and what files are new, and so on. If you create a solid `nneething` group the normal way with *G m*, Gnus will store a mapping table between article numbers and file names, and you can treat this group like any other groups. When you activate a solid `nneething` group, you will be told how many unread articles it contains, etc., etc.

Some variables:

`nneething-map-file-directory`
: All the mapping files for solid `nneething` groups will be stored in this directory, which defaults to '`~/.nneething/`'.

`nneething-exclude-files`
: All files that match this regexp will be ignored. Nice to use to exclude auto-save files and the like, which is what it does by default.

`nneething-map-file`
: Name of the map files.

6.4.3 Document Groups

`nndoc` is a cute little thing that will let you read a single file as a newsgroup. Several files types are supported:

`babyl` The babyl (rmail) mail box.

`mbox` The standard Unix mbox file.

`mmdf` The MMDF mail box format.

`news` Several news articles appended into a file.

`rnews` The rnews batch transport format.

`forward` Forwarded articles.

`mime-digest`
MIME (RFC 1341) digest format.

`standard-digest`
The standard (RFC 1153) digest format.

`slack-digest`
Non-standard digest format—matches most things, but does it badly.

You can also use the special "file type" `guess`, which means that `nndoc` will try to guess what file type it is looking at. `digest` means that `nndoc` should guess what digest type the file is.

`nndoc` will not try to change the file or insert any extra headers into it—it will simply, like, let you use the file as the basis for a group. And that's it.

If you have some old archived articles that you want to insert into your new & spiffy Gnus mail backend, `nndoc` can probably help you with that. Say you have an old '`RMAIL`' file with mail that you now want to split into your new `nnml` groups. You look at that file using `nndoc`, set the process mark on all the articles in the buffer (*M P b*, for instance), and then re-spool (*B r*) using `nnml`. If all goes well, all the mail in the '`RMAIL`' file is now also stored in lots of `nnml` directories, and you can delete that pesky '`RMAIL`' file. If you have the guts!

Virtual server variables:

`nndoc-article-type`
This should be one of `mbox`, `babyl`, `digest`, `mmdf`, `forward`, `news`, `rnews`, `mime-digest`, `clari-briefs`, or `guess`.

`nndoc-post-type`
This variable says whether Gnus is to consider the group a news group or a mail group. There are two legal values: `mail` (the default) and `news`.

6.4.4 SOUP

In the PC world people often talk about "offline" newsreaders. These are thingies that are combined reader/news transport monstrosities. With built-in modem programs. Yecchh!

Of course, us Unix Weenie types of human beans use things like `uucp` and, like, `nntpd` and set up proper news and mail transport things like Ghod intended. And then we just use normal newsreaders.

However, it can sometimes be convenient to do something a that's a bit easier on the brain if you have a very slow modem, and you're not really that interested in doing things properly.

A file format called SOUP has been developed for transporting news and mail from servers to home machines and back again. It can be a bit fiddly.

1. You log in on the server and create a SOUP packet. You can either use a dedicated SOUP thingie, or you can use Gnus to create the packet with the *O s* command.
2. You transfer the packet home. Rail, boat, car or modem will do fine.
3. You put the packet in your home directory.
4. You fire up Gnus using the `nnsoup` backend as the native server.
5. You read articles and mail and answer and followup to the things you want.
6. You do the *G s r* command to pack these replies into a SOUP packet.
7. You transfer this packet to the server.
8. You use Gnus to mail this packet out with the *G s s* command.
9. You then repeat until you die.

So you basically have a bipartite system—you use `nnsoup` for reading and Gnus for packing/sending these SOUP packets.

6.4.4.1 SOUP Commands

G s b Pack all unread articles in the current group (`gnus-group-brew-soup`). This command understands the process/prefix convention.

G s w Save all data files (`gnus-soup-save-areas`).

G s s Send all replies from the replies packet (`gnus-soup-send-replies`).

G s p Pack all files into a SOUP packet (`gnus-soup-pack-packet`).

G s r Pack all replies into a replies packet (`nnsoup-pack-replies`).

O s This summary-mode command adds the current article to a SOUP packet (`gnus-soup-add-article`). It understands the process/prefix convention.

There are a few variables to customize where Gnus will put all these thingies:

`gnus-soup-directory`
: Directory where Gnus will save intermediate files while composing SOUP packets. The default is '`~/SoupBrew/`'.

`gnus-soup-replies-directory`
: This is what Gnus will use as a temporary directory while sending our reply packets. The default is '`~/SoupBrew/SoupReplies/`'.

`gnus-soup-prefix-file`
: Name of the file where Gnus stores the last used prefix. The default is '`gnus-prefix`'.

`gnus-soup-packer`
: A format string command for packing a SOUP packet. The default is '`tar cf - %s | gzip > $HOME/Soupout%d.tgz`'.

`gnus-soup-unpacker`
: Format string command for unpacking a SOUP packet. The default is '`gunzip -c %s | tar xvf -`'.

`gnus-soup-packet-directory`
: Where Gnus will look for reply packets. The default is '`~/`'.

`gnus-soup-packet-regexp`
: Regular expression matching SOUP reply packets in `gnus-soup-packet-directory`.

6.4.4.2 SOUP Groups

`nnsoup` is the backend for reading SOUP packets. It will read incoming packets, unpack them, and put them in a directory where you can read them at leisure.

These are the variables you can use to customize its behavior:

`nnsoup-tmp-directory`
: When `nnsoup` unpacks a SOUP packet, it does it in this directory. ('`/tmp/`' by default.)

`nnsoup-directory`
: `nnsoup` then moves each message and index file to this directory. The default is '`~/SOUP/`'.

`nnsoup-replies-directory`
: All replies will stored in this directory before being packed into a reply packet. The default is '`~/SOUP/replies/"`'.

`nnsoup-replies-format-type`
: The SOUP format of the replies packets. The default is '`?n`' (rnews), and I don't think you should touch that variable. I probably shouldn't even have documented it. Drats! Too late!

`nnsoup-replies-index-type`
: The index type of the replies packet. The is '`?n`', which means "none". Don't fiddle with this one either!

`nnsoup-active-file`
: Where `nnsoup` stores lots of information. This is not an "active file" in the `nntp` sense; it's an Emacs Lisp file. If you lose this file or mess it up in any way, you're dead. The default is '`~/SOUP/active`'.

`nnsoup-packer`
: Format string command for packing a reply SOUP packet. The default is '`tar cf - %s | gzip > $HOME/Soupin%d.tgz`'.

`nnsoup-unpacker`
: Format string command for unpacking incoming SOUP packets. The default is '`gunzip -c %s | tar xvf -`'.

`nnsoup-packet-directory`
: Where `nnsoup` will look for incoming packets. The default is '`~/`'.

`nnsoup-packet-regexp`
: Regular expression matching incoming SOUP packets. The default is '`Soupout`'.

6.4.4.3 SOUP Replies

Just using `nnsoup` won't mean that your postings and mailings end up in SOUP reply packets automagically. You have to work a bit more for that to happen.

The `nnsoup-set-variables` command will set the appropriate variables to ensure that all your followups and replies end up in the SOUP system.

In specific, this is what it does:

```
(setq gnus-inews-article-function 'nnsoup-request-post)
(setq send-mail-function 'nnsoup-request-mail)
```

And that's it, really. If you only want news to go into the SOUP system you just use the first line. If you only want mail to be SOUPed you use the second.

6.5 Combined Groups

Gnus allows combining a mixture of all the other group types into bigger groups.

6.5.1 Virtual Groups

An *nnvirtual group* is really nothing more than a collection of other groups.

For instance, if you are tired of reading many small group, you can put them all in one big group, and then grow tired of reading one big, unwieldy group. The joys of computing!

You specify `nnvirtual` as the method. The address should be a regexp to match component groups.

All marks in the virtual group will stick to the articles in the component groups. So if you tick an article in a virtual group, the article will also be ticked in the component group from whence it came. (And vice versa—marks from the component groups will also be shown in the virtual group.)

Here's an example `nnvirtual` method that collects all Andrea Dworkin newsgroups into one, big, happy newsgroup:

```
(nnvirtual "^alt\\.fan\\.andrea-dworkin$\\|^rec\\.dworkin.*")
```

The component groups can be native or foreign; everything should work smoothly, but if your computer explodes, it was probably my fault.

Collecting the same group from several servers might actually be a good idea if users have set the Distribution header to limit distribution. If you would like to read '`soc.motss`' both from a server in Japan and a server in Norway, you could use the following as the group regexp:

```
"^nntp+some.server.jp:soc.motss$\\|^nntp+some.server.no:soc.motss$"
```

This should work kinda smoothly—all articles from both groups should end up in this one, and there should be no duplicates. Threading (and the rest) will still work as usual, but there might be problems with the sequence of articles. Sorting on date might be an option here (see Section 2.3 [Selecting a Group], page 14.

One limitation, however—all groups that are included in a virtual group has to be alive (i.e., subscribed or unsubscribed). Killed or zombie groups can't be component groups for `nnvirtual` groups.

If the `nnvirtual-always-rescan` is non-`nil`, `nnvirtual` will always scan groups for unread articles when entering a virtual group. If this variable is `nil` (which is the default) and you read articles in a component group after the virtual group has been activated, the read articles from the component group will show up when you enter the virtual group. You'll also see this effect if you have two virtual groups that contain the same component group. If that's the case, you should set this variable to `t`. Or you can just tap `M-g` on the virtual group every time before you enter it—it'll have much the same effect.

6.5.2 Kibozed Groups

Kibozing is defined by OED as "grepping through (parts of) the news feed". `nnkiboze` is a backend that will do this for you. Oh joy! Now you can grind any NNTP server down to a halt with useless requests! Oh happiness!

To create a kibozed group, use the *`G k`* command in the group buffer.

The address field of the `nnkiboze` method is, as with `nnvirtual`, a regexp to match groups to be "included" in the `nnkiboze` group. There most similarities between `nnkiboze` and `nnvirtual` ends.

In addition to this regexp detailing component groups, an `nnkiboze` group must have a score file to say what articles that are to be included in the group (see Chapter 7 [Scoring], page 103).

You must run *`M-x nnkiboze-generate-groups`* after creating the `nnkiboze` groups you want to have. This command will take time. Lots of time. Oodles and oodles of time. Gnus has to fetch the headers from all the articles in all the components groups and run them through the scoring process to determine if there are any articles in the groups that are to be part of the `nnkiboze` groups.

Please limit the number of component groups by using restrictive regexps. Otherwise your sysadmin may become annoyed with you, and the NNTP site may throw you off and never let you back in again. Stranger things have happened.

`nnkiboze` component groups do not have to be alive—they can be dead, and they can be foreign. No restrictions.

The generation of an `nnkiboze` group means writing two files in `nnkiboze-directory`, which is '`~/News/`' by default. One contains the NOV header lines for all the articles in the group, and the other is an additional '`.newsrc`' file to store information on what groups that have been searched through to find component articles.

Articles that are marked as read in the `nnkiboze` group will have their NOV lines removed from the NOV file.

7 Scoring

Other people use *kill files*, but we here at Gnus Towers like scoring better than killing, so we'd rather switch than fight. They do something completely different as well, so sit up straight and pay attention!

All articles have a default score (`gnus-summary-default-score`), which is 0 by default. This score may be raised or lowered either interactively or by score files. Articles that have a score lower than `gnus-summary-mark-below` are marked as read.

Gnus will read any *score files* that apply to the current group before generating the summary buffer.

There are several commands in the summary buffer that insert score entries based on the current article. You can, for instance, ask Gnus to lower or increase the score of all articles with a certain subject.

There are two sorts of scoring entries: Permanent and temporary. Temporary score entries are self-expiring entries. Any entries that are temporary and have not been used for, say, a week, will be removed silently to help keep the sizes of the score files down.

7.1 Summary Score Commands

The score commands that alter score entries do not actually modify real score files. That would be too inefficient. Gnus maintains a cache of previously loaded score files, one of which is considered the *current score file alist*. The score commands simply insert entries into this list, and upon group exit, this list is saved.

The current score file is by default the group's local score file, even if no such score file actually exists. To insert score commands into some other score file (eg. '`all.SCORE`'), you must first make this score file the current one.

General score commands that don't actually change the score file:

`V s` Set the score of the current article (`gnus-summary-set-score`).

`V S` Display the score of the current article (`gnus-summary-current-score`).

`V t` Display all score rules that have been used on the current article (`gnus-score-find-trace`).

`V R` Run the current summary through the scoring process (`gnus-summary-rescore`). This might be useful if you're playing around with your score files behind Gnus' back and want to see the effect you're having.

`V a` Add a new score entry, and allow specifying all elements (`gnus-summary-score-entry`).

`V c` Make a different score file the current (`gnus-score-change-score-file`).

`V e` Edit the current score file (`gnus-score-edit-current-scores`). You will be popped into a `gnus-score-mode` buffer (see Section 7.5 [Score File Editing], page 111).

`V f` Edit a score file and make this score file the current one (`gnus-score-edit-file`).

`V F` Flush the score cahe (`gnus-score-flush-cache`). This is useful after editing score files.

`V C` Customize a score file in a visually pleasing manner (`gnus-score-customize`).

`I C-i` Increase the score of the current article (`gnus-summary-raise-score`).

`L C-l` Lower the score of the current article (`gnus-summary-lower-score`).

The rest of these commands modify the local score file.

`V m` Prompt for a score, and mark all articles with a score below this as read (`gnus-score-set-mark-below`).

`V E` Expunge all articles with a score below the default score (or the numeric prefix) (`gnus-score-set-expunge-below`).

The keystrokes for actually making score entries follow a very regular pattern, so there's no need to list all the commands. (Hundreds of them.)

1. The first key is either `I` (upper case i) for increasing the score or `L` for lowering the score.
2. The second key says what header you want to score on. The following keys are available:

   `a` Score on the author name.

   `s` Score on the subject line.

   `x` Score on the Xref line—i.e., the cross-posting line.

   `t` Score on thread—the References line.

   `d` Score on the date.

   `l` Score on the number of lines.

   `i` Score on the Message-ID.

   `f` Score on followups.

   `b` Score on the body.

   `h` Score on the head.

3. The third key is the match type. Which match types are legal depends on what headers you are scoring on.

   `strings`

   `e` Exact matching.

   `s` Substring matching.

   `f` Fuzzy matching.

   `r` Regexp matching

   `date`

`b` Before date.

`a` At date.

`n` This date.

`number`

`<` Less than number.

`=` Equal to number.

`>` Greater than number.

4. The fourth and final key says whether this is a temporary (i.e., expiring) score entry, or a permanent (i.e., non-expiring) score entry, or whether it is to be done immediately, without adding to the score file.

   `t` Temporary score entry.

   `p` Permanent score entry.

   `i` Immediately scoring.

So, let's say you want to increase the score on the current author with exact matching permanently: `I a e p`. If you want to lower the score based on the subject line, using substring matching, and make a temporary score entry: `L s s t`. Pretty easy.

To make things a bit more complicated, there are shortcuts. If you use a capital letter on either the second or third keys, Gnus will use defaults for the remaining one or two keystrokes. The defaults are "substring" and "temporary". So `I A` is the same as `I a s t`, and `I a R` is the same as `I a r t`.

The `gnus-score-mimic-keymap` says whether these commands will pretend they are keymaps or not.

7.2 Group Score Commands

There aren't many of these as yet, I'm afraid.

`W f` Gnus maintains a cache of score alists to avoid having to reload them all the time. This command will flush the cache (`gnus-score-flush-cache`).

7.3 Score Variables

`gnus-use-scoring`
: If `nil`, Gnus will not check for score files, and will not, in general, do any score-related work. This is `t` by default.

`gnus-kill-killed`
: If this variable is `nil`, Gnus will never apply score files to articles that have already been through the kill process. While this may save you lots of time, it also means that if you apply a kill file to a group, and then change the kill file

and want to run it over you group again to kill more articles, it won't work. You have to set this variable to `t` to do that. (It is `t` by default.)

`gnus-kill-files-directory`
: All kill and score files will be stored in this directory, which is initialized from the `SAVEDIR` environment variable by default. This is '`~/News/`' by default.

`gnus-score-file-suffix`
: Suffix to add to the group name to arrive at the score file name ('`SCORE`' by default.)

`gnus-score-uncacheable-files`
: All score files are normally cached to avoid excessive re-loading of score files. However, if this might make you Emacs grow big and bloated, so this regexp can be used to weed out score files that are unlikely to be needed again. It would be a bad idea to deny caching of '`all.SCORE`', while it might be a good idea to not cache '`comp.infosystems.www.authoring.misc.ADAPT`'. In fact, this variable is '`ADAPT$`' by default, so no adaptive score files will be cached.

`gnus-save-score`
: If you have really complicated score files, and do lots of batch scoring, then you might set this variable to `t`. This will make Gnus save the scores into the '`.newsrc.eld`' file.

`gnus-score-interactive-default-score`
: Score used by all the interactive raise/lower commands to raise/lower score with. Default is 1000, which may seem excessive, but this is to ensure that the adaptive scoring scheme gets enough room to play with. We don't want the small changes from the adaptive scoring to overwrite manually entered data.

`gnus-summary-default-score`
: Default score of an article, which is 0 by default.

`gnus-score-over-mark`
: Mark (in the third column) used for articles with a score over the default. Default is '`+`'.

`gnus-score-below-mark`
: Mark (in the third column) used for articles with a score below the default. Default is '`-`'.

`gnus-score-find-score-files-function`
: Function used to find score files for the current group. This function is called with the name of the group as the argument.

  Predefined functions available are:

  `gnus-score-find-single`
  : Only apply the group's own score file.

  `gnus-score-find-bnews`
  : Apply all score files that match, using bnews syntax. This is the default. For instance, if the current group is '`gnu.emacs.gnus`',

'`all.emacs.all.SCORE`', '`not.alt.all.SCORE`' and '`gnu.all.SCORE`' would all apply. In short, the instances of '`all`' in the score file names are translated into '`.*`', and then a regexp match is done.

This means that if you have some score entries that you want to apply to all groups, then you put those entries in the '`all.SCORE`' file.

`gnus-score-find-hierarchical`
: Apply all score files from all the parent groups. This means that you can't have score files like '`all.SCORE`' or '`all.emacs.SCORE`', but you can have '`SCORE`', '`comp.SCORE`' and '`comp.emacs.SCORE`'.

This variable can also be a list of functions. In that case, all these functions will be called, and all the returned lists of score files will be applied. These functions can also return lists of score alists directly. In that case, the functions that return these non-file score alists should probably be placed before the "real" score file functions, to ensure that the last score file returned is the local score file. Phu.

`gnus-score-expiry-days`
: This variable says how many days should pass before an unused score file entry is expired. If this variable is `nil`, no score file entries are expired. It's 7 by default.

`gnus-update-score-entry-dates`
: If this variable is non-`nil`, matching score entries will have their dates updated. (This is how Gnus controls expiry—all non-matching entries will become too old while matching entries will stay fresh and young.) However, if you set this variable to `nil`, even matching entries will grow old and will have to face that oh-so grim reaper.

`gnus-score-after-write-file-function`
: Function called with the name of the score file just written.

7.4 Score File Format

A score file is an `emacs-lisp` file that normally contains just a single form. Casual users are not expected to edit these files; everything can be changed from the summary buffer.

Anyway, if you'd like to dig into it yourself, here's an example:

```
(("from"
  ("Lars Ingebrigtsen" -10000)
  ("Per Abrahamsen")
  ("larsi\\|lmi" -50000 nil R))
 ("subject"
  ("Ding is Badd" nil 728373))
 ("xref"
  ("alt.politics" -1000 728372 s))
 ("lines"
```

```
 (2 -100 nil <))
(mark 0)
(expunge -1000)
(mark-and-expunge -10)
(read-only nil)
(orphan -10)
(adapt t)
(files "/hom/larsi/News/gnu.SCORE")
(exclude-files "all.SCORE")
(local (gnus-newsgroup-auto-expire t)
       (gnus-summary-make-false-root 'empty))
(eval (ding)))
```

This example demonstrates absolutely everything about a score file.

Even though this looks much like lisp code, nothing here is actually `eval`ed. The lisp reader is used to read this form, though, so it has to be legal syntactically, if not semantically.

Six keys are supported by this alist:

`STRING` If the key is a string, it is the name of the header to perform the match on. Scoring can only be performed on these eight headers: `From`, `Subject`, `References`, `Message-ID`, `Xref`, `Lines`, `Chars` and `Date`. In addition to these headers, there are three strings to tell Gnus to fetch the entire article and do the match on larger parts of the article: `Body` will perform the match on the body of the article, `Head` will perform the match on the head of the article, and `All` will perform the match on the entire article. Note that using any of these last three keys will slow down group entry *considerably*. The final "header" you can score on is `Followup`. These score entries will result in new score entries being added for all follow-ups to articles that matches these score entries.

Following this key is a arbitrary number of score entries, where each score entry has one to four elements.

1. The first element is the *match element*. On most headers this will be a string, but on the Lines and Chars headers, this must be an integer.
2. If the second element is present, it should be a number—the *score element*. This number should be an integer in the neginf to posinf interval. This number is added to the score of the article if the match is successful. If this element is not present, the `gnus-score-interactive-default-score` number will be used instead. This is 1000 by default.
3. If the third element is present, it should be a number—the *date element*. This date says when the last time this score entry matched, which provides a mechanism for expiring the score entries. It this element is not present, the score entry is permanent. The date is represented by the number of days since December 31, 1 ce.
4. If the fourth element is present, it should be a symbol—the *type element*. This element specifies what function should be used to see whether this score entry matches the article. What match types that can be used depends on what header you wish to perform the match on.

From, Subject, References, Xref, Message-ID
: For most header types, there are the `r` and `R` (regexp) as well as `s` and `S` (substring) types and `e` and `E` (exact match) types. If this element is not present, Gnus will assume that substring matching should be used. `R` and `S` differ from the other two in that the matches will be done in a case-sensitive manner. All these one-letter types are really just abbreviations for the `regexp`, `string` and `exact` types, which you can use instead, if you feel like.

Lines, Chars
: These two headers use different match types: `<`, `>`, `=`, `>=` and `<=`.

Date
: For the Date header we have three match types: `before`, `at` and `after`. I can't really imagine this ever being useful, but, like, it would feel kinda silly not to provide this function. Just in case. You never know. Better safe than sorry. Once burnt, twice shy. Don't judge a book by its cover. Never not have sex on a first date.

Head, Body, All
: These three match keys use the same match types as the `From` (etc) header uses.

Followup
: This match key will add a score entry on all articles that followup to some author. Uses the same match types as the `From` header uses.

Thread
: This match key will add a score entry on all articles that are part of a thread. Uses the same match types as the `References` header uses.

`mark`
: The value of this entry should be a number. Any articles with a score lower than this number will be marked as read.

`expunge`
: The value of this entry should be a number. Any articles with a score lower than this number will be removed from the summary buffer.

`mark-and-expunge`
: The value of this entry should be a number. Any articles with a score lower than this number will be marked as read and removed from the summary buffer.

`thread-mark-and-expunge`
: The value of this entry should be a number. All articles that belong to a thread that has a total score below this number will be marked as read and removed from the summary buffer. `gnus-thread-score-function` says how to compute the total score for a thread.

`files`
: The value of this entry should be any number of file names. These files are assumed to be score files as well, and will be loaded the same way this one was.

`exclude-files`
: The clue of this entry should be any number of files. This files will not be loaded, even though they would normally be so, for some reason or other.

`eval`
: The value of this entry will be `eval`el. This element will be ignored when handling global score files.

`read-only`
: Read-only score files will not be updated or saved. Global score files should feature this atom (see Section 7.10 [Global Score Files], page 114).

`orphan`
: The value of this entry should be a number. Articles that do not have parents will get this number added to their scores. Imagine you follow some high-volume newsgroup, like '`comp.lang.c`'. Most likely you will only follow a few of the threads, also want to see any new threads.

  You can do this with the following two score file entries:

  ```
  (orphan -500)
  (mark-and-expunge -100)
  ```

  When you enter the group the first time, you will only see the new threads. You then raise the score of the threads that you find interesting (with *I T* or *I S*), and ignore (*C y*) the rest. Next time you enter the group, you will see new articles in the interesting threads, plus any new threads.

  I.e. – the orphan score atom is for high-volume groups where there exist a few interesting threads which can't be found automatically by ordinary scoring rules.

`adapt`
: This entry controls the adaptive scoring. If it is `t`, the default adaptive scoring rules will be used. If it is `ignore`, no adaptive scoring will be performed on this group. If it is a list, this list will be used as the adaptive scoring rules. If it isn't present, or is something other than `t` or `ignore`, the default adaptive scoring rules will be used. If you want to use adaptive scoring on most groups, you'd set `gnus-use-adaptive-scoring` to `t`, and insert an `(adapt ignore)` in the groups where you do not want adaptive scoring. If you only want adaptive scoring in a few groups, you'd set `gnus-use-adaptive-scoring` to `nil`, and insert `(adapt t)` in the score files of the groups where you want it.

`adapt-file`
: All adaptive score entries will go to the file named by this entry. It will also be applied when entering the group. This atom might be handy if you want to adapt on several groups at once, using the same adaptive file for a number of groups.

`local`
: The value of this entry should be a list of `(VAR VALUE)` pairs. Each *var* will be made buffer-local to the current summary buffer, and set to the value specified. This is a convenient, if somewhat strange, way of setting variables in some groups if you don't like hooks much.

7.5 Score File Editing

You normally enter all scoring commands from the summary buffer, but you might feel the urge to edit them by hand as well, so we've supplied you with a mode for that.

It's simply a slightly customized `emacs-lisp` mode, with these additional commands:

`C-c C-c` Save the changes you have made and return to the summary buffer (`gnus-score-edit-done`).

`C-c C-d` Insert the current date in numerical format (`gnus-score-edit-insert-date`). This is really the day number, if you were wondering.

`C-c C-p` The adaptive score files are saved in an unformatted fashion. If you intend to read one of these files, you want to *pretty print* it first. This command (`gnus-score-pretty-print`) does that for you.

Type `M-x gnus-score-mode` to use this mode.

`gnus-score-menu-hook` is run in score mode buffers.

In the summary buffer you can use commands like `V f` and `V e` to begin editing score files.

7.6 Adaptive Scoring

If all this scoring is getting you down, Gnus has a way of making it all happen automatically—as if by magic. Or rather, as if by artificial stupidity, to be precise.

When you read an article, or mark an article as read, or kill an article, you leave marks behind. On exit from the group, Gnus can sniff these marks and add score elements depending on what marks it finds. You turn on this ability by setting `gnus-use-adaptive-scoring` to `t`.

To give you complete control over the scoring process, you can customize the `gnus-default-adaptive-score-alist` variable. For instance, it might look something like this:

```
(defvar gnus-default-adaptive-score-alist
  '((gnus-unread-mark)
    (gnus-ticked-mark (from 4))
    (gnus-dormant-mark (from 5))
    (gnus-del-mark (from -4) (subject -1))
    (gnus-read-mark (from 4) (subject 2))
    (gnus-expirable-mark (from -1) (subject -1))
    (gnus-killed-mark (from -1) (subject -3))
    (gnus-kill-file-mark)
    (gnus-ancient-mark)
    (gnus-low-score-mark)
    (gnus-catchup-mark (from -1) (subject -1))))
```

As you see, each element in this alist has a mark as a key (either a variable name or a "real" mark—a character). Following this key is a arbitrary number of header/score pairs. If there are no header/score pairs following the key, no adaptive scoring will be done on articles that have that key as the article mark. For instance, articles with `gnus-unread-mark` in the example above will not get adaptive score entries.

Each article can have only one mark, so just a single of these rules will be applied to each article.

To take `gnus-del-mark` as an example—this alist says that all articles that have that mark (i.e., are marked with '`D`') will have a score entry added to lower based on the `From` header by -4, and lowered by `Subject` by -1. Change this to fit your prejudices.

If you have marked 10 articles with the same subject with `gnus-del-mark`, the rule for that mark will be applied ten times. That means that that subject will get a score of ten times -1, which should be, unless I'm much mistaken, -10.

The headers you can score on are `from`, `subject`, `message-id`, `references`, `xref`, `lines`, `chars` and `date`. In addition, you can score on `followup`, which will create an adaptive score entry that matches on the `References` header using the `Message-ID` of the current article, thereby matching the following thread.

You can also score on `thread`, which will try to score all articles that appear in a thread. `thread` matches uses a `Message-ID` to match on the `References` header of the article. If the match is made, the `Message-ID` of the article is added to the `thread` rule. (Think about it. I'd recommend two aspirins afterwards.)

If you use this scheme, you should set the score file atom `mark` to something small—like -300, perhaps, to avoid having small random changes result in articles getting marked as read.

After using adaptive scoring for a week or so, Gnus should start to become properly trained and enhance the authors you like best, and kill the authors you like least, without you having to say so explicitly.

You can control what groups the adaptive scoring is to be performed on by using the score files (see Section 7.4 [Score File Format], page 107). This will also let you use different rules in different groups.

The adaptive score entries will be put into a file where the name is the group name with `gnus-adaptive-file-suffix` appended. The default is '`ADAPT`'.

When doing adaptive scoring, substring or fuzzy matching would probably give you the best results in most cases. However, if the header one matches is short, the possibility for false positives is great, so if the length of the match is less than `gnus-score-exact-adapt-limit`, exact matching will be used. If this variable is `nil`, exact matching will always be used to avoid this problem.

7.7 Followups To Yourself

Gnus offers two commands for picking out the `Message-ID` header in the current buffer. Gnus will then add a score rule that scores using this `Message-ID` on the `References` header of other articles. This will, in effect, increase the score of all articles that respond to the article in the current buffer. Quite useful if you want to easily note when people answer what you've said.

`gnus-score-followup-article`

This will add a score to articles that directly follow up your own article.

`gnus-score-followup-thread`
: This will add a score to all articles that appear in a thread "below" your own article.

These two functions are both primarily meant to be used in hooks like `gnus-inews-article-hook`.

7.8 Scoring Tips

Crossposts
: If you want to lower the score of crossposts, the line to match on is the `Xref` header.

```
("xref" (" talk.politics.misc:" -1000))
```

Multiple crossposts
: If you want to lower the score of articles that have been crossposted to more than, say, 3 groups:

```
("xref" ("[^:\n]+:[0-9]+ +[^:\n]+:[0-9]+ +[^:\n]+:[0-9]+" -1000 nil r))
```

Matching on the body
: This is generally not a very good idea—it takes a very long time. Gnus actually has to fetch each individual article from the server. But you might want to anyway, I guess. Even though there are three match keys (`Head`, `Body` and `All`), you should choose one and stick with it in each score file. If you use any two, each article will be fetched *twice*. If you want to match a bit on the `Head` and a bit on the `Body`, just use `All` for all the matches.

Marking as read
: You will probably want to mark articles that has a score below a certain number as read. This is most easily achieved by putting the following in your '`all.SCORE`' file:

```
((mark -100))
```

You may also consider doing something similar with `expunge`.

Negated character classes
: If you say stuff like `[^abcd]*`, you may get unexpected results. That will match newlines, which might lead to, well, The Unknown. Say `[^abcd\n]*` instead.

7.9 Reverse Scoring

If you want to keep just articles that have '`Sex with Emacs`' in the subject header, and expunge all other articles, you could put something like this in your score file:

```
(("subject"
  ("Sex with Emacs" 2))
 (mark 1)
 (expunge 1))
```

So, you raise all articles that match '`Sex with Emacs`' and mark the rest as read, and expunge them to boot.

7.10 Global Score Files

Sure, other newsreaders have "global kill files". These are usually nothing more than a single kill file that applies to all groups, stored in the user's home directory. Bah! Puny, weak newsreaders!

What I'm talking about here are Global Score Files. Score files from all over the world, from users everywhere, uniting all nations in one big, happy score file union! Ange-score! New and untested!

All you have to do to use other people's score files is to set the `gnus-global-score-files` variable. One entry for each score file, or each score file directory. Gnus will decide by itself what score files are applicable to which group.

Say you want to use all score files in the '`/ftp@ftp.some-where:/pub/score`' directory and the single score file '`/ftp@ftp.ifi.uio.no:/pub/larsi/ding/score/soc.motss.SCORE`':

```
(setq gnus-global-score-files
      '("/ftp@ftp.ifi.uio.no:/pub/larsi/ding/score/soc.motss.SCORE"
        "/ftp@ftp.some-where:/pub/score/"))
```

Simple, eh? Directory names must end with a '`/`'. These directories are typically scanned only once during each Gnus session. If you feel the need to manually re-scan the remote directories, you can use the `gnus-score-search-global-directories` command.

Note that, at present, using this option will slow down group entry somewhat. (That is—a lot.)

If you want to start maintaining score files for other people to use, just put your score file up for anonymous ftp and announce it to the world. Become a retro-moderator! Participate in the retro-moderator wars sure to ensue, where retro-moderators battle it out for the sympathy of the people, luring them to use their score files on false premises! Yay! The net is saved!

Here are some tips for the would-be retro-moderator, off the top of my head:

- Articles that are heavily crossposted are probably junk.
- To lower a single inappropriate article, lower by `Message-ID`.
- Particularly brilliant authors can be raised on a permanent basis.
- Authors that repeatedly post off-charter for the group can safely be lowered out of existence.
- Set the `mark` and `expunge` atoms to obliterate the nastiest articles completely.
- Use expiring score entries to keep the size of the file down. You should probably have a long expiry period, though, as some sites keep old articles for a long time.

... I wonder whether other newsreaders will support global score files in the future. *Snicker*. Yup, any day now, newsreaders like Blue Wave, xrn and 1stReader are bound to implement scoring. Should we start holding our breath yet?

7.11 Kill Files

Gnus still supports those pesky old kill files. In fact, the kill file entries can now be expiring, which is something I wrote before Daniel Quinlan thought of doing score files, so I've left the code in there.

In short, kill processing is a lot slower (and I do mean *a lot*) than score processing, so it might be a good idea to rewrite your kill files into score files.

Anyway, a kill file is a normal `emacs-lisp` file. You can put any forms into this file, which means that you can use kill files as some sort of primitive hook function to be run on group entry, even though that isn't a very good idea.

XCNormal kill files look like this:

```
(gnus-kill "From" "Lars Ingebrigtsen")
(gnus-kill "Subject" "ding")
(gnus-expunge "X")
```

This will mark every article written by me as read, and remove them from the summary buffer. Very useful, you'll agree.

Other programs use a totally different kill file syntax. If Gnus encounters what looks like a `rn` kill file, it will take a stab at interpreting it.

Two summary functions for editing a GNUS kill file:

M-k Edit this group's kill file (`gnus-summary-edit-local-kill`).

M-K Edit the general kill file (`gnus-summary-edit-global-kill`).

Two group mode functions for editing the kill files:

M-k Edit this group's kill file (`gnus-group-edit-local-kill`).

M-K Edit the general kill file (`gnus-group-edit-global-kill`).

Kill file variables:

`gnus-kill-file-name`
: A kill file for the group '`soc.motss`' is normally called '`soc.motss.KILL`'. The suffix appended to the group name to get this file name is detailed by the `gnus-kill-file-name` variable. The "global" kill file (not in the score file sense of "global", of course) is called just '`KILL`'.

`gnus-kill-save-kill-file`
: If this variable is non-`nil`, Gnus will save the kill file after processing, which is necessary if you use expiring kills.

`gnus-apply-kill-hook`
: A hook called to apply kill files to a group. It is `(gnus-apply-kill-file)` by default. If you want to ignore the kill file if you have a score file for the same group, you can set this hook to `(gnus-apply-kill-file-unless-scored)`. If you don't want kill files to be processed, you should set this variable to `nil`.

`gnus-kill-file-mode-hook`
: A hook called in kill-file mode buffers.

7.12 GroupLens

GroupLens is a collaborative filtering system that helps you work together with other people to find the quality news articles out of the huge volume of news articles generated every day.

To accomplish this the GroupLens system combines your opinions about articles you have already read with the opinions of others who have done likewise and gives you a personalized prediction for each unread news article. Think of GroupLens as a matchmaker. GroupLens watches how you rate articles, and finds other people that rate articles the same way. Once it has found for you some people you agree with it tells you, in the form of a prediction, what they thought of the article. You can use this prediction to help you decide whether or not you want to read the article.

7.12.1 Using GroupLens

To use GroupLens you must register a pseudonym with your local Better Bit Bureau (BBB). At the moment the only better bit in town is at `http://www.cs.umn.edu/Research/GroupLens/bbb.h`

Once you have registered you'll need to set a couple of variables.

`gnus-use-grouplens`
: Setting this variable to a non-`nil` value will make Gnus hook into all the relevant GroupLens functions.

`grouplens-pseudonym`
: This variable should be set to the pseudonum you got when registering with the Better Bit Bureau.

`grouplens-newsgroups`
: A list of groups that you want to get GroupLens predictions for.

Thats the minimum of what you need to get up and running with GroupLens. Once you've registered, GroupLens will start giving you scores for articles based on the average of what other people think. But, to get the real benefit of GroupLens you need to start rating articles yourself. Then the scores GroupLens gives you will be personalized for you, based on how the people you usually agree with have already rated.

7.12.2 Rating Articles

In GroupLens, an article is rated on a scale from 1 to 5, inclusive. Where 1 means something like this article is a waste of bandwidth and 5 means that the article was really good. The basic question to ask yourself is, "on a scale from 1 to 5 would I like to see more articles like this one?"

There are four ways to enter a rating for an article in GroupLens.

r
: This function will prompt you for a rating on a scale of one to five.

k This function will prompt you for a rating, and rate all the articles in the thread. This is really useful for some of those long running giant threads in rec.humor.

The next two commands, *n* and *,* take a numerical prefix to be the score of the article you're reading.

1-5 n Rate the article and go to the next unread article.

1-5 , Rate the article and go to the next unread article with the highest score.

If you want to give the current article a score of 4 and then go to the next article, just type *4 n*.

7.12.3 Displaying Predictions

GroupLens makes a prediction for you about how much you will like a news article. The predictions from GroupLens are on a scale from 1 to 5, where 1 is the worst and 5 is the best. You can use the predictions from GroupLens in one of three ways controlled by the variable `gnus-grouplens-override-scoring`.

There are three ways to display predictions in grouplens. You may choose to have the GroupLens scores contribute to, or override the regular gnus scoring mechanism. override is the default; however, some people prefer to see the Gnus scores plus the grouplens scores. To get the separate scoring behavior you need to set `gnus-grouplens-override-scoring` to `'separate`. To have the GroupLens predictions combined with the grouplens scores set it to `'override` and to combine the scores set `gnus-grouplens-override-scoring` to `'combine`. When you use the combine option you will also want to set the values for `grouplens-prediction-offset` and `grouplens-score-scale-factor`.

In either case, GroupLens gives you a few choices for how you would like to see your predictions displayed. The display of predictions is controlled by the `grouplens-prediction-display` variable.

The following are legal values for that variable.

`prediction-spot`
: The higher the prediction, the further to the right an '`*`' is displayed.

`confidence-interval`
: A numeric confidence interval.

`prediction-bar`
: The higher the prediction, the longer the bar.

`confidence-bar`
: Numerical confidence.

`confidence-spot`
: The spot gets bigger with more confidence.

`prediction-num`
: Plain-old numeric value.

`confidence-plus-minus`
: Prediction +/i confidence.

7.12.4 GroupLens Variables

`gnus-summary-grouplens-line-format`
: The summary line format used in summary buffers that are GroupLens enhanced. It accepts the same specs as the normal summary line format (see Section 3.1.1 [Summary Buffer Lines], page 29). The default is '`%U%R%z%l%I%(%[%4L: %-20,20n%]%) %s\n`'.

`grouplens-bbb-host`
: Host running the bbbd server. The default is '`grouplens.cs.umn.edu`'.

`grouplens-bbb-port`
: Port of the host running the bbbd server. The default is 9000.

`grouplens-score-offset`
: Offset the prediction by this value. In other words, subtract the prediction value by this number to arrive at the effective score. The default is 0.

`grouplens-score-scale-factor`
: This variable allows the user to magnify the effect of GroupLens scores. The scale factor is applied after the offset. The default is 1.

8 Various

8.1 Process/Prefix

Many functions, among them functions for moving, decoding and saving articles, use what is known as the *Process/Prefix convention*.

This is a method for figuring out what articles that the user wants the command to be performed on.

It goes like this:

If the numeric prefix is N, perform the operation on the next N articles, starting with the current one. If the numeric prefix is negative, perform the operation on the previous N articles, starting with the current one.

If `transient-mark-mode` in non-`nil` and the region is active, all articles in the region will be worked upon.

If there is no numeric prefix, but some articles are marked with the process mark, perform the operation on the articles that are marked with the process mark.

If there is neither a numeric prefix nor any articles marked with the process mark, just perform the operation on the current article.

Quite simple, really, but it needs to be made clear so that surprises are avoided.

One thing that seems to shock & horrify lots of people is that, for instance, `3 d` does exactly the same as `d d d`. Since each `d` (which marks the current article as read) by default goes to the next unread article after marking, this means that `3 d` will mark the next three unread articles as read, no matter what the summary buffer looks like. Set `gnus-summary-goto-unread` to `nil` for a more straightforward action.

8.2 Interactive

`gnus-novice-user`
: If this variable is non-`nil`, you are either a newcomer to the World of Usenet, or you are very cautious, which is a nice thing to be, really. You will be given questions of the type "Are you sure you want to do this?" before doing anything dangerous. This is `t` by default.

`gnus-expert-user`
: If this variable is non-`nil`, you will never ever be asked any questions by Gnus. It will simply assume you know what you're doing, no matter how strange.

`gnus-interactive-catchup`
: Require confirmation before catching up a group if non-`nil`. It is `t` by default.

`gnus-interactive-exit`
: Require confirmation before exiting Gnus. This variable is `t` by default.

8.3 Formatting Variables

Throughout this manual you've probably noticed lots of variables that are called things like `gnus-group-line-format` and `gnus-summary-mode-line-format`. These control how Gnus is to output lines in the various buffers. There's quite a lot of them. Fortunately, they all use the same syntax, so there's not that much to be annoyed by.

Here's an example format spec (from the group buffer): '`%M%S%5y: %(%g%)\n`'. We see that it is indeed extremely ugly, and that there are lots of percentages everywhere.

Each '`%`' element will be replaced by some string or other when the buffer in question is generated. '`%5y`' means "insert the '`y`' spec, and pad with spaces to get a 5-character field". Just like a normal format spec, almost.

You can also say '`%6,4y`', which means that the field will never be more than 6 characters wide and never less than 4 characters wide.

There are also specs for highlighting, and these are shared by all the format variables. Text inside the '`%(`' and '`%)`' specifiers will get the special `mouse-face` property set, which means that it will be highlighted (with `gnus-mouse-face`) when you put the mouse pointer over it.

Text inside the '`%[`' and '`%]`' specifiers will have their normal faces set using `gnus-face-0`, which is `bold` by default. If you say '`%1[`' instead, you'll get `gnus-face-1` instead, and so on. Create as many faces as you wish. The same goes for the `mouse-face` specs—you can say '`%3(hello%)`' to have '`hello`' mouse-highlighted with `gnus-mouse-face-3`.

Here's an alternative recipe for the group buffer:

```
;; Create three face types.
(setq gnus-face-1 'bold)
(setq gnus-face-3 'italic)

;; We want the article count to be in
;; a bold and green face.  So we create
;; a new face called `my-green-bold'.
(copy-face 'bold 'my-green-bold)
;; Set the color.
(set-face-foreground 'my-green-bold "ForestGreen")
(setq gnus-face-2 'my-green-bold)

;; Set the new & fancy format.
(setq gnus-group-line-format
      "%M%S%3{%5y%}%2[:%] %(%1{%g%}%)\n")
```

I'm sure you'll be able to use this scheme to create totally unreadable and extremely vulgar displays. Have fun!

Currently Gnus uses the following formatting variables: `gnus-group-line-format`, `gnus-summary-line-format`, `gnus-server-line-format`, `gnus-topic-line-format`, `gnus-group-mode-line-format`, `gnus-summary-mode-line-format`, `gnus-article-mode-line-format`, `gnus-server-mode-line-format`.

Note that the '`%(`' specs (and friends) do not make any sense on the mode-line variables.

All these format variables can also be arbitrary elisp forms. In that case, they will be `eval`ed to insert the required lines.

Gnus includes a command to help you while creating your own format specs. *M-x gnus-update-format* will `eval` the current form, update the spec in question and pop you to a buffer where you can examine the resulting lisp code to be run to generate the line.

8.4 Windows Configuration

No, there's nothing here about X, so be quiet.

If `gnus-use-full-window` non-`nil`, Gnus will delete all other windows and occupy the entire Emacs screen by itself. It is `t` by default.

`gnus-buffer-configuration` describes how much space each Gnus buffer should be given. Here's an excerpt of this variable:

```
((group (vertical 1.0 (group 1.0 point)
                      (if gnus-carpal (group-carpal 4))))
 (article (vertical 1.0 (summary 0.25 point)
                        (article 1.0))))
```

This is an alist. The *key* is a symbol that names some action or other. For instance, when displaying the group buffer, the window configuration function will use `group` as the key. A full list of possible names is listed below.

The *value* (i. e., the *split*) says how much space each buffer should occupy. To take the `article` split as an example -

```
(article (vertical 1.0 (summary 0.25 point)
                       (article 1.0)))
```

This *split* says that the summary buffer should occupy 25% of upper half of the screen, and that it is placed over the article buffer. As you may have noticed, 100% + 25% is actually 125% (yup, I saw y'all reaching for that calculator there). However, the special number `1.0` is used to signal that this buffer should soak up all the rest of the space available after the rest of the buffers have taken whatever they need. There should be only one buffer with the `1.0` size spec per split.

Point will be put in the buffer that has the optional third element `point`.

Here's a more complicated example:

```
(article (vertical 1.0 (group 4)
                       (summary 0.25 point)
                       (if gnus-carpal (summary-carpal 4))
                       (article 1.0)))
```

If the size spec is an integer instead of a floating point number, then that number will be used to say how many lines a buffer should occupy, not a percentage.

If the *split* looks like something that can be `eval`ed (to be precise—if the `car` of the split is a function or a subr), this split will be `eval`ed. If the result is non-`nil`, it will be used as a split. This means that there will be three buffers if `gnus-carpal` is `nil`, and four buffers if `gnus-carpal` is non-`nil`.

Not complicated enough for you? Well, try this on for size:

```
(article (horizontal 1.0
             (vertical 0.5
                 (group 1.0)
                 (gnus-carpal 4))
             (vertical 1.0
                 (summary 0.25 point)
                 (summary-carpal 4)
                 (article 1.0))))
```

Whoops. Two buffers with the mystery 100% tag. And what's that `horizontal` thingie?

If the first element in one of the split is `horizontal`, Gnus will split the window horizontally, giving you two windows side-by-side. Inside each of these strips you may carry on all you like in the normal fashion. The number following `horizontal` says what percentage of the screen is to be given to this strip.

For each split, there *must* be one element that has the 100% tag. The splitting is never accurate, and this buffer will eat any leftover lines from the splits.

To be slightly more formal, here's a definition of what a legal split may look like:

```
split       = frame | horizontal | vertical | buffer | form
frame       = "(frame " size *split ")"
horizontal  = "(horizontal " size *split ")"
vertical    = "(vertical " size *split ")"
buffer      = "(" buffer-name " " size *[ "point" ] ")"
size        = number | frame-params
buffer-name = group | article | summary ...
```

The limitations are that the `frame` split can only appear as the top-level split. *form* should be an Emacs Lisp form that should return a valid split. We see that each split is fully recursive, and may contain any number of `vertical` and `horizontal` splits.

Finding the right sizes can be a bit complicated. No window may be less than `gnus-window-min-height` (default 2) characters high, and all windows must be at least `gnus-window-min-width` (default 1) characters wide. Gnus will try to enforce this before applying the splits. If you want to use the normal Emacs window width/height limit, you can just set these two variables to `nil`.

If you're not familiar with Emacs terminology, `horizontal` and `vertical` splits may work the opposite way of what you'd expect. Windows inside a `horizontal` split are shown side-by-side, and windows within a `vertical` split are shown above each other.

If you want to experiment with window placement, a good tip is to call `gnus-configure-frame` directly with a split. This is the function that does all the real work when splitting buffers. Below is a pretty nonsensical configuration with 5 windows; two for the group buffer and three for the article buffer. (I said it was nonsensical.) If you `eval` the statement below, you can get an idea of how that would look straight away, without going through the normal Gnus channels. Play with it until you're satisfied, and then use `gnus-add-configuration` to add your new creation to the buffer configuration list.

```
(gnus-configure-frame
 '(horizontal 1.0
    (vertical 10
      (group 1.0)
```

```
      (article 0.3 point))
    (vertical 1.0
      (article 1.0)
      (horizontal 4
        (group 1.0)
        (article 10)))))
```

You might want to have several frames as well. No prob—just use the `frame` split:

```
(gnus-configure-frame
 '(frame 1.0
         (vertical 1.0
                   (summary 0.25 point)
                   (article 1.0))
         (vertical ((height . 5) (width . 15)
                    (user-position . t)
                    (left . -1) (top . 1))
                   (picon 1.0))))
```

This split will result in the familiar summary/article window configuration in the first (or "main") frame, while a small additional frame will be created where picons will be shown. As you can see, instead of the normal `1.0` top-level spec, each additional split should have a frame parameter alist as the size spec. See section "Frame Parameters" in *The GNU Emacs Lisp Reference Manual*.

Here's a list of all possible keys for `gnus-buffer-configuration`:

`group`, `summary`, `article`, `server`, `browse`, `group-mail`, `summary-mail`, `summary-reply`, `info`, `summary-faq`, `edit-group`, `edit-server`, `reply`, `reply-yank`, `followup`, `followup-yank`, `edit-score`.

Since the `gnus-buffer-configuration` variable is so long and complicated, there's a function you can use to ease changing the config of a single setting: `gnus-add-configuration`. If, for instance, you want to change the `article` setting, you could say:

```
(gnus-add-configuration
 '(article (vertical 1.0
                     (group 4)
                     (summary .25 point)
                     (article 1.0))))
```

You'd typically stick these `gnus-add-configuration` calls in your '`.gnus`' file or in some startup hook – they should be run after Gnus has been loaded.

8.5 Compilation

Remember all those line format specification variables? `gnus-summary-line-format`, `gnus-group-line-format`, and so on. Now, Gnus will of course heed whatever these variables are, but, unfortunately, changing them will mean a quite significant slow-down. (The default values of these variables have byte-compiled functions associated with them, while the user-generated versions do not, of course.)

To help with this, you can run *M-x gnus-compile* after you've fiddled around with the variables and feel that you're (kind of) satisfied. This will result in the new specs being byte-compiled, and you'll get top speed again.

8.6 Mode Lines

`gnus-updated-mode-lines` says what buffers should keep their mode lines updated. It is a list of symbols. Supported symbols include `group`, `article`, `summary`, `server`, `browse`, and `tree`. If the corresponding symbol is present, Gnus will keep that mode line updated with information that may be pertinent. If this variable is `nil`, screen refresh may be quicker.

By default, Gnus displays information on the current article in the mode lines of the summary and article buffers. The information Gnus wishes to display (eg. the subject of the article) is often longer than the mode lines, and therefore have to be cut off at some point. The `gnus-mode-non-string-length` variable says how long the other elements on the line is (i.e., the non-info part). If you put additional elements on the mode line (eg. a clock), you should modify this variable:

```
(add-hook 'display-time-hook
          (lambda () (setq gnus-mode-non-string-length
                           (+ 21
                              (if line-number-mode 5 0)
                              (if column-number-mode 4 0)
                              (length display-time-string)))))
```

If this variable is `nil` (which is the default), the mode line strings won't be chopped off, and they won't be padded either.

8.7 Highlighting and Menus

The `gnus-visual` variable controls most of the prettifying Gnus aspects. If `nil`, Gnus won't attempt to create menus or use fancy colors or fonts. This will also inhibit loading the '`gnus-vis.el`' file.

This variable can be a list of visual properties that are enabled. The following elements are legal, and are all included by default:

`group-highlight`
: Do highlights in the group buffer.

`summary-highlight`
: Do highlights in the summary buffer.

`article-highlight`
: Do highlights in the article buffer.

`highlight`
: Turn on highlighting in all buffers.

`group-menu`
: Create menus in the group buffer.

`summary-menu`
: Create menus in the summary buffers.

`article-menu`
: Create menus in the article buffer.

`browse-menu`
: Create menus in the browse buffer.

`server-menu`
: Create menus in the server buffer.

`score-menu`
: Create menus in the score buffers.

`menu`
: Create menus in all buffers.

So if you only want highlighting in the article buffer and menus in all buffers, you could say something like:

```
(setq gnus-visual '(article-highlight menu))
```

If you want only highlighting and no menus whatsoever, you'd say:

```
(setq gnus-visual '(highlight))
```

If `gnus-visual` is `t`, highlighting and menus will be used in all Gnus buffers.

Other general variables that influence the look of all buffers include:

`gnus-mouse-face`
: This is the face (i.e., font) used for mouse highlighting in Gnus. No mouse highlights will be done if `gnus-visual` is `nil`.

`gnus-display-type`
: This variable is symbol indicating the display type Emacs is running under. The symbol should be one of `color`, `grayscale` or `mono`. If Gnus guesses this display attribute wrongly, either set this variable in your '`~/.emacs`' or set the resource `Emacs.displayType` in your '`~/.Xdefaults`'.

`gnus-background-mode`
: This is a symbol indicating the Emacs background brightness. The symbol should be one of `light` or `dark`. If Gnus guesses this frame attribute wrongly, either set this variable in your '`~/.emacs`' or set the resource `Emacs.backgroundMode` in your '`~/.Xdefaults`'. 'gnus-display-type'.

There are hooks associated with the creation of all the different menus:

`gnus-article-menu-hook`
: Hook called after creating the article mode menu.

`gnus-group-menu-hook`
: Hook called after creating the group mode menu.

`gnus-summary-menu-hook`
: Hook called after creating the summary mode menu.

`gnus-server-menu-hook`
: Hook called after creating the server mode menu.

`gnus-browse-menu-hook`
: Hook called after creating the browse mode menu.

`gnus-score-menu-hook`
: Hook called after creating the score mode menu.

8.8 Buttons

Those new-fangled *mouse* contraptions is very popular with the young, hep kids who don't want to learn the proper way to do things these days. Why, I remember way back in the summer of '89, when I was using Emacs on a Tops 20 system. Three hundred users on one single machine, and every user was running Simula compilers. Bah!

Right.

Well, you can make Gnus display bufferfuls of buttons you can click to do anything by setting `gnus-carpal` to `t`. Pretty simple, really. Tell the chiropractor I sent you.

`gnus-carpal-mode-hook`
: Hook run in all carpal mode buffers.

`gnus-carpal-button-face`
: Face used on buttons.

`gnus-carpal-header-face`
: Face used on carpal buffer headers.

`gnus-carpal-group-buffer-buttons`
: Buttons in the group buffer.

`gnus-carpal-summary-buffer-buttons`
: Buttons in the summary buffer.

`gnus-carpal-server-buffer-buttons`
: Buttons in the server buffer.

`gnus-carpal-browse-buffer-buttons`
: Buttons in the browse buffer.

All the `buttons` variables are lists. The elements in these list is either a cons cell where the car contains a text to be displayed and the cdr contains a function symbol, or a simple string.

8.9 Daemons

Gnus, being larger than any program ever written (allegedly), does lots of strange stuff that you may wish to have done while you're not present. For instance, you may want it to check for new mail once in a while. Or you may want it to close down all connections to all servers when you leave Emacs idle. And stuff like that.

Gnus will let you do stuff like that by defining various *handlers*. Each handler consists of three elements: A *function*, a *time*, and an *idle* parameter.

Here's an example of a handler that closes connections when Emacs has been idle for thirty minutes:

```
(gnus-demon-close-connections nil 30)
```

Here's a handler that scans for PGP headers every hour when Emacs is idle:

```
(gnus-demon-scan-pgp 60 t)
```

This *time* parameter and than *idle* parameter works together in a strange, but wonderful fashion. Basically, if *idle* is `nil`, then the function will be called every *time* minutes.

If *idle* is `t`, then the function will be called after *time* minutes only if Emacs is idle. So if Emacs is never idle, the function will never be called. But once Emacs goes idle, the function will be called every *time* minutes.

If *idle* is a number and *time* is a number, the function will be called every *time* minutes only when Emacs has been idle for *idle* minutes.

If *idle* is a number and *time* is `nil`, the function will be called once every time Emacs has been idle for *idle* minutes.

And if *time* is a string, it should look like '`07:31`', and the function will then be called once every day somewhere near that time. Modified by the *idle* parameter, of course.

(When I say "minute" here, I really mean `gnus-demon-timestep` seconds. This is `60` by default. If you change that variable, all the timings in the handlers will be affected.)

To set the whole thing in motion, though, you have to set `gnus-use-demon` to `t`.

So, if you want to add a handler, you could put something like this in your '`.gnus`' file:

```
(gnus-demon-add-handler 'gnus-demon-close-connections nil 30)
```

Some ready-made functions to do this has been created: `gnus-demon-add-nocem`, `gnus-demon-add-disconnection`, and `gnus-demon-add-scanmail`. Just put those functions in your '`.gnus`' if you want those abilities.

If you add handlers to `gnus-demon-handlers` directly, you should run `gnus-demon-init` to make the changes take hold. To cancel all daemons, you can use the `gnus-demon-cancel` function.

Note that adding daemons can be pretty naughty if you overdo it. Adding functions that scan all news and mail from all servers every two seconds is a sure-fire way of getting booted off any respectable system. So behave.

8.10 NoCeM

Spamming is posting the same article lots and lots of times. Spamming is bad. Spamming is evil.

Spamming is usually canceled within a day or so by various anti-spamming agencies. These agencies usually also send out *NoCeM* messages. NoCeM is pronounced "no see-'em", and means what the name implies—these are messages that make the offending articles, like, go away.

What use are these NoCeM messages if the articles are canceled anyway? Some sites do not honor cancel messages and some sites just honor cancels from a select few people. Then you may wish to make use of the NoCeM messages, which are distributed in the '`alt.nocem.misc`' newsgroup.

Gnus can read and parse the messages in this group automatically, and this will make spam disappear.

There are some variables to customize, of course:

`gnus-use-nocem`
: Set this variable to `t` to set the ball rolling. It is `nil` by default.

`gnus-nocem-groups`
: Gnus will look for NoCeM messages in the groups in this list. The default is `("alt.nocem.misc" "news.admin.net-abuse.announce")`.

`gnus-nocem-issuers`
: There are many people issuing NoCeM messages. This list says what people you want to listen to. The default is `("Automoose-1" "clewis@ferret.ocunix.on.ca;" "jem@xpat.com;" "red@redpoll.mrfs.oh.us (Richard E. Depew)")`; fine, upstanding citizens all of them.

  Known despammers that you can put in this list include:

  '`clewis@ferret.ocunix.on.ca;`'
  : Chris Lewis—Major Canadian despammer who has probably canceled more usenet abuse than anybody else.

  '`Automoose-1`'
  : The CancelMoose[tm] on autopilot. The CancelMoose[tm] is reputed to be Norwegian, and was the person(s) who invented NoCeM.

  '`jem@xpat.com;`'
  : Jem—Korean despammer who is getting very busy these days.

  '`red@redpoll.mrfs.oh.us (Richard E. Depew)`'
  : Richard E. Depew—lone American despammer. He mostly cancels binary postings to non-binary groups and removes spews (regurgitated articles).

  You do not have to heed NoCeM messages from all these people—just the ones you want to listen to.

`gnus-nocem-directory`
: This is where Gnus will store its NoCeM cache files. The default is '`~/News/NoCeM/`'.

`gnus-nocem-expiry-wait`
: The number of days before removing old NoCeM entries from the cache. The default is 15. If you make it shorter Gnus will be faster, but you might then see old spam.

8.11 Picons

So... You want to slow down your news reader even more! This is a good way to do so. Its also a great way to impress people staring over your shoulder as you read news.

8.11.1 Picon Basics

What are Picons? To quote directly from the Picons Web site (‘`http://www.cs.indiana.edu/picons/ftp/`

> *Picons* is short for “personal icons”. They’re small, constrained images used to represent users and domains on the net, organized into databases so that the appropriate image for a given e-mail address can be found. Besides users and domains, there are picon databases for Usenet newsgroups and weather forecasts. The picons are in either monochrome `XBM` format or color `XPM` and `GIF` formats.

Please see the above mentioned web site for instructions on obtaining and installing the picons databases, or the following ftp site: ‘`http://www.cs.indiana.edu/picons/ftp/index.html`’.

Gnus expects picons to be installed into a location pointed to by `gnus-picons-database`.

8.11.2 Picon Requirements

To use have Gnus display Picons for you, you must be running XEmacs 19.13 or greater since all other versions of Emacs aren’t yet able to display images.

Additionally, you must have `xpm` support compiled into XEmacs.

If you want to display faces from `X-Face` headers, you must have the `netpbm` utilities installed, or munge the `gnus-picons-convert-x-face` variable to use something else.

8.11.3 Easy Picons

To enable displaying picons, simply put the following line in your ‘`~/.gnus`’ file and start Gnus.

```
(setq gnus-use-picons t)
(add-hook 'gnus-article-display-hook 'gnus-article-display-picons t)
(add-hook 'gnus-summary-prepare-hook 'gnus-group-display-picons t)
(add-hook 'gnus-article-display-hook 'gnus-picons-article-display-x-face)
```

8.11.4 Hard Picons

Gnus can display picons for you as you enter and leave groups and articles. It knows how to interact with three sections of the picons database. Namely, it can display the picons newsgroup pictures, author’s face picture(s), and the authors domain. To enable this feature, you need to first decide where to display them.

`gnus-picons-display-where`
: Where the picon images should be displayed. It is `picons` by default (which by default maps to the buffer '`*Picons*`'). Other valid places could be `article`, `summary`, or '`"*scratch*"`' for all I care. Just make sure that you've made the buffer visible using the standard Gnus window configuration routines – See Section 8.4 [Windows Configuration], page 121.

Note: If you set `gnus-use-picons` to `t`, it will set up your window configuration for you to include the `picons` buffer.

Now that you've made that decision, you need to add the following functions to the appropriate hooks so these pictures will get displayed at the right time.

`gnus-article-display-picons`
: Looks up and display the picons for the author and the author's domain in the `gnus-picons-display-where` buffer. Should be added to the `gnus-article-display-hook`.

`gnus-group-display-picons`
: Displays picons representing the current group. This function should be added to the `gnus-summary-prepare-hook` or to the `gnus-article-display-hook` if `gnus-picons-display-where` is set to `article`.

`gnus-picons-article-display-x-face`
: Decodes and displays the X-Face header if present. This function should be added to `gnus-article-display-hook`.

Note: You must append them to the hook, so make sure to specify 't' to the append flag of `add-hook`:

```
(add-hook 'gnus-article-display-hook 'gnus-article-display-picons t)
```

8.11.5 Picon Configuration

The following variables offer further control over how things are done, where things are located, and other useless stuff you really don't need to worry about.

`gnus-picons-database`
: The location of the picons database. Should point to a directory containing the '`news`', '`domains`', '`users`' (and so on) subdirectories. Defaults to '`/usr/local/faces`'.

`gnus-picons-news-directory`
: Sub-directory of the faces database containing the icons for newsgroups.

`gnus-picons-user-directories`
: List of subdirectories to search in `gnus-picons-database` for user faces. Defaults to `("local" "users" "usenix" "misc/MISC")`.

`gnus-picons-domain-directories`
: List of subdirectories to search in `gnus-picons-database` for domain name faces. Defaults to `("domains")`. Some people may want to add '`unknown`' to this list.

`gnus-picons-convert-x-face`
: The command to use to convert the `X-Face` header to an X bitmap (`xbm`). Defaults to `(format "{ echo '/* Width=48, Height=48 */'; uncompface; } | icontopbm | pbmtoxbm > %s" gnus-picons-x-face-file-name)`

`gnus-picons-x-face-file-name`
: Names a temporary file to store the `X-Face` bitmap in. Defaults to `(format "/tmp/picon-xface.%s.xbm" (user-login-name))`.

`gnus-picons-buffer`
: The name of the buffer that `picons` points to. Defaults to '`*Icon Buffer*`'.

8.12 Various Various

`gnus-verbose`
: This variable is an integer between zero and ten. The higher the value, the more messages will be displayed. If this variable is zero, Gnus will never flash any messages, if it is seven (which is the default), most important messages will be shown, and if it is ten, Gnus won't ever shut up, but will flash so many messages it will make your head swim.

`gnus-verbose-backends`
: This variable works the same way as `gnus-verbose`, but it applies to the Gnus backends instead of Gnus proper.

`nnheader-max-head-length`
: When the backends read straight heads of articles, they all try to read as little as possible. This variable (default `4096`) specifies the absolute max length the backends will try to read before giving up on finding a separator line between the head and the body. If this variable is `nil`, there is no upper read bound. If it is `t`, the backends won't try to read the articles piece by piece, but read the entire articles. This makes sense with some versions of `ange-ftp`.

`nnheader-file-name-translation-alist`
: This is an alist that says how to translate characters in file names. For instance, if '`:`' is illegal as a file character in file names on your system (you OS/2 user you), you could say something like:

```
(setq nnheader-file-name-translation-alist
      '((?: . ?_)))
```

: In fact, this is the default value for this variable on OS/2 and MS Windows (phooey) systems.

`gnus-hidden-properties`
: This is a list of properties to use to hide "invisible" text. It is `(invisible t intangible t)` by default on most systems, which makes invisible text invisible and intangible.

`gnus-parse-headers-hook`
: A hook called before parsing headers. It can be used, for instance, to gather statistics on the headers fetched, or perhaps you'd like to prune some headers. I don't see why you'd want that, though.

9 The End

Well, that's the manual—you can get on with your life now. Keep in touch. Say hello to your cats from me.

My **ghod**—I just can't stand goodbyes. Sniffle.

Ol' Charles Reznikoff said it pretty well, so I leave the floor to him:

Te Deum

Not because of victories
I sing,
having none,
but for the common sunshine,
the breeze,
the largess of the spring.

Not for victory
but for the day's work done
as well as I was able;
not for a seat upon the dais
but at the common table.

10 Appendices

10.1 History

GNUS was written by Masanobu UMEDA. When autumn crept up in '94, Lars Magne Ingebrigtsen grew bored and decided to rewrite Gnus.

If you want to investigate the person responsible for this outrage, you can point your (feh!) web browser to '`http://www.ifi.uio.no/~larsi/`'. This is also the primary distribution point for the new and spiffy versions of Gnus, and is known as The Site That Destroys Newsrcs And Drives People Mad.

During the first extended alpha period of development, the new Gnus was called "(ding) Gnus". *(ding)*, is, of course, short for *ding is not Gnus*, which is a total and utter lie, but who cares? (Besides, the "Gnus" in this abbreviation should probably be pronounced "news" as UMEDA intended, which makes it a more appropriate name, don't you think?)

In any case, after spending all that energy on coming up with a new and spunky name, we decided that the name was *too* spunky, so we renamed it back again to "Gnus". But in mixed case. "Gnus" vs. "GNUS". New vs. old.

The first "proper" release of Gnus 5 was done in November 1995 when it was included in the Emacs 19.30 distribution.

In May 1996 the next Gnus generation (aka. "September Gnus") was released under the name "Gnus 5.2".

10.1.1 Why?

What's the point of Gnus?

I want to provide a "rad", "happening", "way cool" and "hep" newsreader, that lets you do anything you can think of. That was my original motivation, but while working on Gnus, it has become clear to me that this generation of newsreaders really belong in the stone age. Newsreaders haven't developed much since the infancy of the net. If the volume continues to rise with the current rate of increase, all current newsreaders will be pretty much useless. How do you deal with newsgroups that have thousands of new articles each day? How do you keep track of millions of people who post?

Gnus offers no real solutions to these questions, but I would very much like to see Gnus being used as a testing ground for new methods of reading and fetching news. Expanding on UMEDA-san's wise decision to separate the newsreader from the backends, Gnus now offers a simple interface for anybody who wants to write new backends for fetching mail and news from different sources. I have added hooks for customizations everywhere I could imagine useful. By doing so, I'm inviting every one of you to explore and invent.

May Gnus never be complete. *`C-u 100 M-x hail-emacs`*.

10.1.2 Compatibility

Gnus was designed to be fully compatible with GNUS. Almost all key bindings have been kept. More key bindings have been added, of course, but only in one or two obscure cases have old bindings been changed.

Our motto is:

> In a cloud bones of steel.

All commands have kept their names. Some internal functions have changed their names.

The `gnus-uu` package has changed drastically. see Section 3.16 [Decoding Articles], page 51.

One major compatibility question is the presence of several summary buffers. All variables that are relevant while reading a group are buffer-local to the summary buffer they belong in. Although many important variables have their values copied into their global counterparts whenever a command is executed in the summary buffer, this change might lead to incorrect values being used unless you are careful.

All code that relies on knowledge of GNUS internals will probably fail. To take two examples: Sorting `gnus-newsrc-alist` (or changing it in any way, as a matter of fact) is strictly verboten. Gnus maintains a hash table that points to the entries in this alist (which speeds up many functions), and changing the alist directly will lead to peculiar results.

Old hilit19 code does not work at all. In fact, you should probably remove all hilit code from all Gnus hooks (`gnus-group-prepare-hook` and `gnus-summary-prepare-hook`). Gnus provides various integrated functions for highlighting. These are faster and more accurate. To make life easier for everybody, Gnus will by default remove all hilit calls from all hilit hooks. Uncleanliness! Away!

Packages like `expire-kill` will no longer work. As a matter of fact, you should probably remove all old GNUS packages (and other code) when you start using Gnus. More likely than not, Gnus already does what you have written code to make GNUS do. (Snicker.)

Even though old methods of doing things are still supported, only the new methods are documented in this manual. If you detect a new method of doing something while reading this manual, that does not mean you have to stop doing it the old way.

Gnus understands all GNUS startup files.

Overall, a casual user who hasn't written much code that depends on GNUS internals should suffer no problems. If problems occur, please let me know by issuing that magic command *`M-x gnus-bug`*.

10.1.3 Conformity

No rebels without a clue here, ma'am. We conform to all standards known to (wo)man. Except for those standards and/or conventions we disagree with, of course.

RFC 822 There are no known breaches of this standard.

RFC 1036 There are no known breaches of this standard, either.

Usenet Seal of Approval
Gnus hasn't been formally through the Seal process, but I have read through the Seal text and I think Gnus would pass.

Son-of-RFC 1036
We do have some breaches to this one.

MIME Gnus does no MIME handling, and this standard-to-be seems to think that MIME is the bees' knees, so we have major breakage here.

X-Newsreader
This is considered to be a "vanity header", while I consider it to be consumer information. After seeing so many badly formatted articles coming from `tin` and `Netscape` I know not to use either of those for posting articles. I would not have known that if it wasn't for the `X-Newsreader` header.

References
Gnus does line breaking on this header. I infer from RFC1036 that being conservative in what you output is not creating 5000-character lines, so it seems like a good idea to me. However, this standard-to-be says that whitespace in the `References` header is to be preserved, so... It doesn't matter one way or the other to Gnus, so if somebody tells me what The Way is, I'll change it. Or not.

If you ever notice Gnus acting non-compliantly with regards to the texts mentioned above, don't hesitate to drop a note to Gnus Towers and let us know.

10.1.4 Emacsen

Gnus should work on :

- Emacs 19.30 and up.
- XEmacs 19.13 and up.
- Mule versions based on Emacs 19.30 and up.

Gnus will absolutely not work on any Emacsen older than that. Not reliably, at least.

There are some vague differences between Gnus on the various platforms:

- The mouse-face on Gnus lines under Emacs and Mule is delimited to certain parts of the lines while they cover the entire line under XEmacs.
- The same with current-article marking—XEmacs puts an underline under the entire summary line while Emacs and Mule are nicer and kinder.
- XEmacs features more graphics—a logo and a toolbar.
- Citation highlighting us better under Emacs and Mule than under XEmacs.
- Emacs 19.26-19.28 have tangible hidden headers, which can be a bit confusing.

10.1.5 Contributors

The new Gnus version couldn't have been done without the help of all the people on the (ding) mailing list. Every day for over a year I have gotten billions of nice bug reports from them, filling me with joy, every single one of them. Smooches. The people on the list have been tried beyond endurance, what with my "oh, that's a neat idea <type type>, yup, I'll release it right away <ship off> no wait, that doesn't work at all <type type>, yup, I'll ship that one off right away <ship off> no, wait, that absolutely does not work" policy for releases. Micro$oft—bah. Amateurs. I'm *much* worse. (Or is that "worser"? "much worser"? "worsest"?)

I would like to take this opportunity to thank the Academy for... oops, wrong show.

- Masanobu UMEDA The writer of the original GNUS.
- Per Abrahamsen Custom, scoring, highlighting and SOUP code (as well as numerous other things).
- Luis Fernandes Design and graphics.
- Wes Hardaker '`gnus-picon.el`' and the manual section on *picons* (see Section 8.11 [Picons], page 129).
- Brad Miller '`gnus-gl.el`' and the GroupLens manual section (see Section 7.12 [GroupLens], page 116).
- Sudish Joseph Innumerable bug fixes.
- Ilja Weis '`gnus-topic.el`'.
- Steven L. Baur Lots and lots of bugs detections and fixes.
- Vladimir Alexiev The refcard and reference booklets.
- Felix Lee & JWZ I stole some pieces from the XGnus distribution by Felix Lee and JWZ.
- Scott Byer '`nnfolder.el`' enhancements & rewrite.
- Peter Mutsaers Orphan article scoring code.
- Ken Raeburn POP mail support.
- Hallvard B Furuseth Various bits and pieces, especially dealing with .newsrc files.
- Brian Edmonds '`gnus-bbdb.el`'.
- Ricardo Nassif and Mark Borges Proof-reading.
- Kevin Davidson Came up with the name *ding*, so blame him.

Peter Arius, Stainless Steel Rat, Ulrik Dickow, Jack Vinson, Daniel Quinlan, Frank D. Cringle, Geoffrey T. Dairiki, Fabrice Popineau and Andrew Eskilsson have all contributed code and suggestions.

10.1.6 New Features

- The look of all buffers can be changed by setting format-like variables (see Section 2.1 [Group Buffer Format], page 11 and see Section 3.1 [Summary Buffer Format], page 29).

- Local spool and several NNTP servers can be used at once (see Chapter 6 [Select Methods], page 77).
- You can combine groups into virtual groups (see Section 6.5.1 [Virtual Groups], page 99).
- You can read a number of different mail formats (see Section 6.3 [Getting Mail], page 83). All the mail backends implement a convenient mail expiry scheme (see Section 6.3.7 [Expiring Mail], page 89).
- Gnus can use various strategies for gathering threads that have lost their roots (thereby gathering loose sub-threads into one thread) or it can go back and retrieve enough headers to build a complete thread (see Section 3.9.1 [Customizing Threading], page 41).
- Killed groups can be displayed in the group buffer, and you can read them as well (see Section 2.10 [Listing Groups], page 21).
- Gnus can do partial group updates—you do not have to retrieve the entire active file just to check for new articles in a few groups (see Section 1.9 [The Active File], page 8).
- Gnus implements a sliding scale of subscribedness to groups (see Section 2.5 [Group Levels], page 16).
- You can score articles according to any number of criteria (see Chapter 7 [Scoring], page 103). You can even get Gnus to find out how to score articles for you (see Section 7.6 [Adaptive Scoring], page 111).
- Gnus maintains a dribble buffer that is auto-saved the normal Emacs manner, so it should be difficult to lose much data on what you have read if your machine should go down (see Section 1.8 [Auto Save], page 7).
- Gnus now has its own startup file (‘`.gnus`’) to avoid cluttering up the ‘`.emacs`’ file.
- You can set the process mark on both groups and articles and perform operations on all the marked items (see Section 8.1 [Process/Prefix], page 119).
- You can grep through a subset of groups and create a group from the results (see Section 6.5.2 [Kibozed Groups], page 100).
- You can list subsets of groups according to, well, anything (see Section 2.10 [Listing Groups], page 21).
- You can browse foreign servers and subscribe to groups from those servers (see Section 2.13 [Browse Foreign Server], page 23).
- Gnus can fetch articles asynchronously on a second connection to the server (see Section 3.11 [Asynchronous Fetching], page 46).
- You can cache articles locally (see Section 3.12 [Article Caching], page 46).
- The uudecode functions have been expanded and generalized (see Section 3.16 [Decoding Articles], page 51).
- You can still post uuencoded articles, which was a little-known feature of GNUS’ past (see Section 3.16.4.3 [Uuencoding and Posting], page 54).
- Fetching parents (and other articles) now actually works without glitches (see Section 3.19 [Finding the Parent], page 60).
- Gnus can fetch FAQs and group descriptions (see Section 2.16.2 [Group Information], page 27).

- Digests (and other files) can be used as the basis for groups (see Section 6.4.3 [Document Groups], page 96).
- Articles can be highlighted and customized (see Section 4.3 [Customizing Articles], page 71).
- URLs and other external references can be buttonized (see Section 3.17.4 [Article Buttons], page 58).
- You can do lots of strange stuff with the Gnus window & frame configuration (see Section 8.4 [Windows Configuration], page 121).
- You can click on buttons instead of using the keyboard (see Section 8.8 [Buttons], page 126).
- Gnus can use NoCeM files to weed out spam (see Section 8.10 [NoCeM], page 127).

This is, of course, just a *short* overview of the *most* important new features. No, really. There are tons more. Yes, we have feeping creaturism in full effect, but nothing too gratuitous, I would hope.

10.1.7 Newest Features

Also known as the *todo list*. Sure to be implemented before the next millennium.

Be afraid. Be very afraid.

- Native MIME support is something that should be done.
- A better and simpler method for specifying mail composing methods.
- Allow posting through mail-to-news gateways.
- Really do unbinhexing.

And much, much, much more. There is more to come than has already been implemented. (But that's always true, isn't it?)

`<URL:http://www.ifi.uio.no/~larsi/sgnus/todo>` is where the actual up-to-the-second todo list is located, so if you're really curious, you could point your Web browser over that-a-way.

10.2 Terminology

news This is what you are supposed to use this thing for—reading news. News is generally fetched from a nearby NNTP server, and is generally publicly available to everybody. If you post news, the entire world is likely to read just what you have written, and they'll all snigger mischievously. Behind your back.

mail Everything that's delivered to you personally is mail. Some news/mail readers (like Gnus) blur the distinction between mail and news, but there is a difference. Mail is private. News is public. Mailing is not posting, and replying is not following up.

reply Send a mail to the person who has written what you are reading.

follow up Post an article to the current newsgroup responding to the article you are reading.

backend Gnus gets fed articles from a number of backends, both news and mail backends. Gnus does not handle the underlying media, so to speak—this is all done by the backends.

native Gnus will always use one method (and backend) as the *native*, or default, way of getting news.

foreign You can also have any number of foreign groups active at the same time. These are groups that use different backends for getting news.

secondary Secondary backends are somewhere half-way between being native and being foreign, but they mostly act like they are native.

article A nessage that has been posted as news.

mail message
A message that has been mailed.

message A mail message or news article

head The top part of a message, where administrative information (etc.) is put.

body The rest of an article. Everything that is not in the head is in the body.

header A line from the head of an article.

headers A collection of such lines, or a collection of heads. Or even a collection of NOV lines.

NOV When Gnus enters a group, it asks the backend for the headers of all unread articles in the group. Most servers support the News OverView format, which is more compact and much faster to read and parse than the normal HEAD format.

level Each group is subscribed at some *level* or other (1-9). The ones that have a lower level are "more" subscribed than the groups with a higher level. In fact, groups on levels 1-5 are considered *subscribed*; 6-7 are *unsubscribed*; 8 are *zombies*; and 9 are *killed*. Commands for listing groups and scanning for new articles will all use the numeric prefix as *working level*.

killed groups
No information on killed groups is stored or updated, which makes killed groups much easier to handle than subscribed groups.

zombie groups
Just like killed groups, only slightly less dead.

active file The news server has to keep track of what articles it carries, and what groups exist. All this information in stored in the active file, which is rather large, as you might surmise.

bogus groups
: A group that exists in the '`.newsrc`' file, but isn't known to the server (i. e., it isn't in the active file), is a *bogus group*. This means that the group probably doesn't exist (any more).

server
: A machine than one can connect to and get news (or mail) from.

select method
: A structure that specifies the backend, the server and the virtual server parameters.

virtual server
: A named select method. Since a select methods defines all there is to know about connecting to a (physical) server, taking the who things as a whole is a virtual server.

10.3 Customization

All variables are properly documented elsewhere in this manual. This section is designed to give general pointers on how to customize Gnus for some quite common situations.

10.3.1 Slow/Expensive NNTP Connection

If you run Emacs on a machine locally, and get your news from a machine over some very thin strings, you want to cut down on the amount of data Gnus has to get from the NNTP server.

`gnus-read-active-file`
: Set this to `nil`, which will inhibit Gnus from requesting the entire active file from the server. This file is often v. large. You also have to set `gnus-check-new-news` and `gnus-check-bogus-newsgroups` to `nil` to make sure that Gnus doesn't suddenly decide to fetch the active file anyway.

`gnus-nov-is-evil`
: This one has to be `nil`. If not, grabbing article headers from the NNTP server will not be very fast. Not all NNTP servers support XOVER; Gnus will detect this by itself.

10.3.2 Slow Terminal Connection

Let's say you use your home computer for dialing up the system that runs Emacs and Gnus. If your modem is slow, you want to reduce the amount of data that is sent over the wires as much as possible.

`gnus-auto-center-summary`
: Set this to `nil` to inhibit Gnus from re-centering the summary buffer all the time. If it is `vertical`, do only vertical re-centering. If it is neither `nil` nor `vertical`, do both horizontal and vertical recentering.

`gnus-visible-headers`
: Cut down on the headers that are included in the articles to the minimum. You can, in fact, make do without them altogether—most of the useful data is in the summary buffer, anyway. Set this variable to '`^NEVVVVER`' or '`From:`', or whatever you feel you need.

`gnus-article-display-hook`
: Set this hook to all the available hiding commands:

```
(setq gnus-article-display-hook
      '(gnus-article-hide-headers gnus-article-hide-signature
        gnus-article-hide-citation))
```

`gnus-use-full-window`
: By setting this to `nil`, you can make all the windows smaller. While this doesn't really cut down much generally, it means that you have to see smaller portions of articles before deciding that you didn't want to read them anyway.

`gnus-thread-hide-subtree`
: If this is non-`nil`, all threads in the summary buffer will be hidden initially.

`gnus-updated-mode-lines`
: If this is `nil`, Gnus will not put information in the buffer mode lines, which might save some time.

10.3.3 Little Disk Space

The startup files can get rather large, so you may want to cut their sizes a bit if you are running out of space.

`gnus-save-newsrc-file`
: If this is `nil`, Gnus will never save '`.newsrc`'—it will only save '`.newsrc.eld`'. This means that you will not be able to use any other newsreaders than Gnus. This variable is `t` by default.

`gnus-save-killed-list`
: If this is `nil`, Gnus will not save the list of dead groups. You should also set `gnus-check-new-newsgroups` to `ask-server` and `gnus-check-bogus-newsgroups` to `nil` if you set this variable to `nil`. This variable is `t` by default.

10.3.4 Slow Machine

If you have a slow machine, or are just really impatient, there are a few things you can do to make Gnus run faster.

Set`gnus-check-new-newsgroups` and `gnus-check-bogus-newsgroups` to `nil` to make startup faster.

Set `gnus-show-threads`, `gnus-use-cross-reference` and `gnus-nov-is-evil` to `nil` to make entering and exiting the summary buffer faster.

Set `gnus-article-display-hook` to `nil` to make article processing a bit faster.

10.4 Troubleshooting

Gnus works *so* well straight out of the box—I can't imagine any problems, really.

Ahem.

1. Make sure your computer is switched on.
2. Make sure that you really load the current Gnus version. If you have been running GNUS, you need to exit Emacs and start it up again before Gnus will work.
3. Try doing an *M-x gnus-version*. If you get something that looks like '`Gnus v5.46; nntp 4.0`' you have the right files loaded. If, on the other hand, you get something like '`NNTP 3.x`' or '`nntp flee`', you have some old '`.el`' files lying around. Delete these.
4. Read the help group (*G h* in the group buffer) for a FAQ and a how-to.

If all else fails, report the problem as a bug.

If you find a bug in Gnus, you can report it with the *M-x gnus-bug* command. *M-x set-variable RET debug-on-error RET t RET*, and send me the backtrace. I will fix bugs, but I can only fix them if you send me a precise description as to how to reproduce the bug.

You really can never be too detailed in a bug report. Always use the *M-x gnus-bug* command when you make bug reports, even if it creates a 10Kb mail each time you use it, and even if you have sent me your environment 500 times before. I don't care. I want the full info each time.

It is also important to remember that I have no memory whatsoever. If you send a bug report, and I send you a reply, and then you send back just "No, it's not! Moron!", I will have no idea what you are insulting me about. Always over-explain everything. It's much easier for all of us—if I don't have all the information I need, I will just mail you and ask for more info, and everything takes more time.

If the problem you're seeing is very visual, and you can't quite explain it, copy the Emacs window to a file (with `xwd`, for instance), put it somewhere it can be reached, and include the URL of the picture in the bug report.a

If you just need help, you are better off asking on '`gnu.emacs.gnus`'. I'm not very helpful.

You can also ask on the ding mailing list—'`ding@ifi.uio.no`'. Write to '`ding-request@ifi.uio.no`' to subscribe.

10.5 A Programmer's Guide to Gnus

It is my hope that other people will figure out smart stuff that Gnus can do, and that other people will write those smart things as well. To facilitate that I thought it would be a good idea to describe the inner workings of Gnus. And some of the not-so-inner workings, while I'm at it.

You can never expect the internals of a program not to change, but I will be defining (in some details) the interface between Gnus and its backends (this is written in stone), the format of the score files (ditto), data structures (some are less likely to change than others) and general method of operations.

10.5.1 Backend Interface

Gnus doesn't know anything about NNTP, spools, mail or virtual groups. It only knows how to talk to *virtual servers*. A virtual server is a *backend* and some *backend variables*. As examples of the first, we have `nntp`, `nnspool` and `nnmbox`. As examples of the latter we have `nntp-port-number` and `nnmbox-directory`.

When Gnus asks for information from a backend—say `nntp`—on something, it will normally include a virtual server name in the function parameters. (If not, the backend should use the "current" virtual server.) For instance, `nntp-request-list` takes a virtual server as its only (optional) parameter. If this virtual server hasn't been opened, the function should fail.

Note that a virtual server name has no relation to some physical server name. Take this example:

```
(nntp "odd-one"
      (nntp-address "ifi.uio.no")
      (nntp-port-number 4324))
```

Here the virtual server name is '`odd-one`' while the name of the physical server is '`ifi.uio.no`'.

The backends should be able to switch between several virtual servers. The standard backends implement this by keeping an alist of virtual server environments that it pulls down/pushes up when needed.

There are two groups of interface functions: *required functions*, which must be present, and *optional functions*, which Gnus will always check whether are present before attempting to call.

All these functions are expected to return data in the buffer `nntp-server-buffer` ('`*nntpd*`'), which is somewhat unfortunately named, but we'll have to live with it. When I talk about "resulting data", I always refer to the data in that buffer. When I talk about "return value", I talk about the function value returned by the function call.

Some backends could be said to be *server-forming* backends, and some might be said to not be. The latter are backends that generally only operate on one group at a time, and have no concept of "server" – they have a group, and they deliver info on that group and nothing more.

In the examples and definitions I will refer to the imaginary backend `nnchoke`.

10.5.1.1 Required Backend Functions

`(nnchoke-retrieve-headers ARTICLES &optional GROUP SERVER FETCH-OLD)`

> *articles* is either a range of article numbers or a list of `Message-ID`s. Current backends do not fully support either—only sequences (lists) of article numbers, and most backends do not support retrieval of `Message-ID`s. But they should try for both.
>
> The result data should either be HEADs or NOV lines, and the result value should either be `headers` or `nov` to reflect this. This might later be expanded

to `various`, which will be a mixture of HEADs and NOV lines, but this is currently not supported by Gnus.

If *fetch-old* is non-`nil` it says to try to fetch "extra headers, in some meaning of the word. This is generally done by fetching (at most) *fetch-old* extra headers less than the smallest article number in `articles`, and fill in the gaps as well. The presence of this parameter can be ignored if the backend finds it cumbersome to follow the request. If this is non-`nil` and not a number, do maximum fetches.

Here's an example HEAD:

```
221 1056 Article retrieved.
Path: ifi.uio.no!sturles
From: sturles@ifi.uio.no (Sturle Sunde)
Newsgroups: ifi.discussion
Subject: Re: Something very droll
Date: 27 Oct 1994 14:02:57 +0100
Organization: Dept. of Informatics, University of Oslo, Norway
Lines: 26
Message-ID: <38o8e1$a0o@holmenkollen.ifi.uio.no>
References: <38jdmq$4qu@visbur.ifi.uio.no>
NNTP-Posting-Host: holmenkollen.ifi.uio.no
.
```

So a `headers` return value would imply that there's a number of these in the data buffer.

Here's a BNF definition of such a buffer:

```
headers        = *head
head           = error / valid-head
error-message  = [ "4" / "5" ] 2number " " <error message> eol
valid-head     = valid-message *header "." eol
valid-message  = "221 " <number> " Article retrieved." eol
header         = <text> eol
```

If the return value is `nov`, the data buffer should contain *network overview database* lines. These are basically fields separated by tabs.

```
nov-buffer = *nov-line
nov-line   = 8*9 [ field <TAB> ] eol
field      = <text except TAB>
```

For a closer explanation what should be in those fields, see Section 10.5.3 [Headers], page 155.

`(nnchoke-open-server SERVER &optional DEFINITIONS)`

server is here the virtual server name. *definitions* is a list of `(VARIABLE VALUE)` pairs that defines this virtual server.

If the server can't be opened, no error should be signaled. The backend may then choose to refuse further attempts at connecting to this server. In fact, it should do so.

If the server is opened already, this function should return a non-`nil` value. There should be no data returned.

`(nnchoke-close-server &optional SERVER)`
: Close connection to *server* and free all resources connected to it. Return `nil` if the server couldn't be closed for some reason.

  There should be no data returned.

`(nnchoke-request-close)`
: Close connection to all servers and free all resources that the backend have reserved. All buffers that have been created by that backend should be killed. (Not the `nntp-server-buffer`, though.) This function is generally only called when Gnus is shutting down.

  There should be no data returned.

`(nnchoke-server-opened &optional SERVER)`
: If *server* is the current virtual server, and the connection to the physical server is alive, then this function should return a non-`nil` vlue. This function should under no circumstances attempt to reconnect to a server that is has lost connection to.

  There should be no data returned.

`(nnchoke-status-message &optional SERVER)`
: This function should return the last error message from *server*.

  There should be no data returned.

`(nnchoke-request-article ARTICLE &optional GROUP SERVER TO-BUFFER)`
: The result data from this function should be the article specified by *article*. This might either be a `Message-ID` or a number. It is optional whether to implement retrieval by `Message-ID`, but it would be nice if that were possible.

  If *to-buffer* is non-`nil`, the result data should be returned in this buffer instead of the normal data buffer. This is to make it possible to avoid copying large amounts of data from one buffer to another, and Gnus mainly request articles to be inserted directly into its article buffer.

  If it is at all possible, this function should return a cons cell where the car is the group name the article was fetched from, and the cdr is the article number. This will enable Gnus to find out what the real group and article numbers are when fetching articles by `Message-ID`. If this isn't possible, `t` should be returned on successful article retrievement.

`(nnchoke-open-group GROUP &optional SERVER)`
: Make *group* the current group.

  There should be no data returned by this function.

`(nnchoke-request-group GROUP &optional SERVER)`
: Get data on *group*. This function also has the side effect of making *group* the current group.

  Here's an example of some result data and a definition of the same:

  ```
  211 56 1000 1059 ifi.discussion
  ```

  The first number is the status, which should be `211`. Next is the total number of articles in the group, the lowest article number, the highest article number,

and finally the group name. Note that the total number of articles may be less than one might think while just considering the highest and lowest article numbers, but some articles may have been canceled. Gnus just discards the total-number, so whether one should take the bother to generate it properly (if that is a problem) is left as an exercise to the reader.

```
group-status = [ error / info ] eol
error        = [ "4" / "5" ] 2<number> " " <Error message>
info         = "211 " 3* [ <number> " " ] <string>
```

`(nnchoke-close-group GROUP &optional SERVER)`

Close *group* and free any resources connected to it. This will be a no-op on most backends.

There should be no data returned.

`(nnchoke-request-list &optional SERVER)`

Return a list of all groups available on *server*. And that means *all*.

Here's an example from a server that only carries two groups:

```
ifi.test 0000002200 0000002000 y
ifi.discussion 3324 3300 n
```

On each line we have a group name, then the highest article number in that group, the lowest article number, and finally a flag.

```
active-file = *active-line
active-line = name " " <number> " " <number> " " flags eol
name        = <string>
flags       = "n" / "y" / "m" / "x" / "j" / "=" name
```

The flag says whether the group is read-only (‘`n`’), is moderated (‘`m`’), is dead (‘`x`’), is aliased to some other group (‘`=other-group`’ or none of the above (‘`y`’).

`(nnchoke-request-post &optional SERVER)`

This function should post the current buffer. It might return whether the posting was successful or not, but that's not required. If, for instance, the posting is done asynchronously, it has generally not been completed by the time this function concludes. In that case, this function should set up some kind of sentinel to beep the user loud and clear if the posting could not be completed.

There should be no result data from this function.

10.5.1.2 Optional Backend Functions

`(nnchoke-retrieve-groups GROUPS &optional SERVER)`

groups is a list of groups, and this function should request data on all those groups. How it does it is of no concern to Gnus, but it should attempt to do this in a speedy fashion.

The return value of this function can be either `active` or `group`, which says what the format of the result data is. The former is in the same format as the

data from `nnchoke-request-list`, while the latter is a buffer full of lines in the same format as `nnchoke-request-group` gives.

```
group-buffer = *active-line / *group-status
```

`(nnchoke-request-update-info GROUP INFO &optional SERVER)`

A Gnus group info (see Section 10.5.5 [Group Info], page 156) is handed to the backend for alterations. This comes in handy if the backend really carries all the information (as is the case with virtual an imap groups). This function may alter the info in any manner it sees fit, and should return the (altered) group info. This function may alter the group info destructively, so no copying is needed before boogeying.

There should be no result data from this function.

`(nnchoke-request-type GROUP &optional ARTICLE)`

When the user issues commands for "sending news" (`F` in the summary buffer, for instance), Gnus has to know whether the article the user is following up is news or mail. This function should return `news` if *article* in *group* is news, `mail` if it is mail and `unknown` if the type can't be decided. (The *article* parameter is necessary in `nnvirtual` groups which might very well combine mail groups and news groups.)

There should be no result data from this function.

`(nnchoke-request-update-mark GROUP ARTICLE MARK)`

If the user tries to set a mark that the backend doesn't like, this function may change the mark. Gnus will use whatever this function returns as the mark for *article* instead of the original *mark*. If the backend doesn't care, it must return the original *mark*, and not `nil` or any other type of garbage.

The only use for this that I can see is what `nnvirtual` does with it—if a component group is auto-expirable, marking an article as read in the virtual group should result in the article being marked as expirable.

There should be no result data from this function.

`(nnchoke-request-scan &optional GROUP SERVER)`

This function may be called at any time (by Gnus or anything else) to request that the backend check for incoming articles, in one way or another. A mail backend will typically read the spool file or query the POP server when this function is invoked. The *group* doesn't have to be heeded—if the backend decides that it is too much work just scanning for a single group, it may do a total scan of all groups. It would be nice, however, to keep things local if that's practical.

There should be no result data from this function.

`(nnchoke-request-asynchronous GROUP &optional SERVER ARTICLES)`

This is a request to fetch articles asynchronously later. *articles* is an alist of *(article-number line-number)*. One would generally expect that if one later fetches article number 4, for instance, some sort of asynchronous fetching of the articles after 4 (which might be 5, 6, 7 or 11, 3, 909 depending on the order in

that alist) would be fetched asynchronously, but that is left up to the backend. Gnus doesn't care.

There should be no result data from this function.

`(nnchoke-request-group-description GROUP &optional SERVER)`

The result data from this function should be a description of *group*.

```
description-line = name <TAB> description eol
name             = <string>
description      = <text>
```

`(nnchoke-request-list-newsgroups &optional SERVER)`

The result data from this function should be the description of all groups available on the server.

```
description-buffer = *description-line
```

`(nnchoke-request-newgroups DATE &optional SERVER)`

The result data from this function should be all groups that were created after '`date`', which is in normal human-readable date format. The data should be in the active buffer format.

`(nnchoke-request-create-group GROUP &optional SERVER)`

This function should create an empty group with name *group*.

There should be no return data.

`(nnchoke-request-expire-articles ARTICLES &optional GROUP SERVER FORCE)`

This function should run the expiry process on all articles in the *articles* range (which is currently a simple list of article numbers.) It is left up to the backend to decide how old articles should be before they are removed by this function. If *force* is non-`nil`, all *articles* should be deleted, no matter how new they are.

This function should return a list of articles that it did not/was not able to delete.

There should be no result data returned.

`(nnchoke-request-move-article ARTICLE GROUP SERVER ACCEPT-FORM &optional LAST)`

This function should move *article* (which is a number) from *group* by calling *accept-form*.

This function should ready the article in question for moving by removing any header lines it has added to the article, and generally should "tidy up" the article. Then it should `eval` *accept-form* in the buffer where the "tidy" article is. This will do the actual copying. If this `eval` returns a non-`nil` value, the article should be removed.

If *last* is `nil`, that means that there is a high likelihood that there will be more requests issued shortly, so that allows some optimizations.

The function should return a cons where the car is the group name and the cdr is the article number that the article was entered as.

There should be no data returned.

`(nnchoke-request-accept-article GROUP &optional SERVER LAST)`
: This function takes the current buffer and inserts it into *group*. If *last* in `nil`, that means that there will be more calls to this function in short order.

  The function should return a cons where the car is the group name and the cdr is the article number that the article was entered as.

  There should be no data returned.

`(nnchoke-request-replace-article ARTICLE GROUP BUFFER)`
: This function should remove *article* (which is a number) from *group* and insert *buffer* there instead.

  There should be no data returned.

`(nnchoke-request-delete-group GROUP FORCE &optional SERVER)`
: This function should delete *group*. If *force*, it should really delete all the articles in the group, and then delete the group itself. (If there is such a thing as "the group itself".)

  There should be no data returned.

`(nnchoke-request-rename-group GROUP NEW-NAME &optional SERVER)`
: This function should rename *group* into *new-name*. All articles that are in *group* should move to *new-name*.

  There should be no data returned.

10.5.1.3 Writing New Backends

The various backends share many similarities. `nnml` is just like `nnspool`, but it allows you to edit the articles on the server. `nnmh` is just like `nnml`, but it doesn't use an active file, and it doesn't maintain overview databases. `nndir` is just like `nnml`, but it has no concept of "groups", and it doesn't allow editing articles.

It would make sense if it were possible to "inherit" functions from backends when writing new backends. And, indeed, you can do that if you want to. (You don't have to if you don't want to, of course.)

All the backends declare their public variables and functions by using a package called `nnoo`.

To inherit functions from other backends (and allow other backends to inherit functions from the current backend), you should use the following macros: following.

`nnoo-declare`
: This macro declares the first parameter to be a child of the subsequent parameters. For instance:

  ```
  (nnoo-declare nndir
    nnml nnmh)
  ```

  `nndir` has here declared that it intends to inherit functions from both `nnml` and `nnmh`.

`defvoo` This macro is equivalent to `defvar`, but registers the variable as a public server variable. Most state-oriented variables should be declared with `defvoo` instead of `defvar`.

In addition to the normal `defvar` parameters, it takes a list of variables in the parent backends to map the variable to when executing a function in those backends.

```
(defvoo nndir-directory nil
  "Where nndir will look for groups."
  nnml-current-directory nnmh-current-directory)
```

This means that `nnml-current-directory` will be set to `nndir-directory` when an `nnml` function is called on behalf of `nndir`. (The same with `nnmh`.)

`nnoo-define-basics`
This macro defines some common functions that almost all backends should have.

```
(nnoo-define-basics nndir)
```

`deffoo` This macro is just like `defun` and takes the same parameters. In addition to doing the normal `defun` things, it registers the function as being public so that other backends can inherit it.

`nnoo-map-functions`
This macro allows mapping of functions from the current backend to functions from the parent backends.

```
(nnoo-map-functions nndir
  (nnml-retrieve-headers 0 nndir-current-group 0 0)
  (nnmh-request-article 0 nndir-current-group 0 0))
```

This means that when `nndir-retrieve-headers` is called, the first, third, and fourth parameters will be passed on to `nnml-retrieve-headers`, while the second parameter is set to the value of `nndir-current-group`.

`nnoo-import`
This macro allows importing functions from backends. It should be the last thing in the source file, since it will only define functions that haven't already been defined.

```
(nnoo-import nndir
  (nnmh
   nnmh-request-list
   nnmh-request-newgroups)
  (nnml))
```

This means that calls to `nndir-request-list` should just be passed on to `nnmh-request-list`, while all public functions from `nnml` that haven't been defined in `nndir` yet should be defined now.

Below is a slightly shortened version of the `nndir` backend.

```
;;; nndir.el --- single directory newsgroup access for Gnus
;; Copyright (C) 1995,96 Free Software Foundation, Inc.
```

```
;;; Code:

(require 'nnheader)
(require 'nnmh)
(require 'nnml)
(require 'nnoo)
(eval-when-compile (require 'cl))

(nnoo-declare nndir
  nnml nnmh)

(defvoo nndir-directory nil
  "Where nndir will look for groups."
  nnml-current-directory nnmh-current-directory)

(defvoo nndir-nov-is-evil nil
  "*Non-nil means that nndir will never retrieve NOV headers."
  nnml-nov-is-evil)

(defvoo nndir-current-group "" nil nnml-current-group nnmh-current-group)
(defvoo nndir-top-directory nil nil nnml-directory nnmh-directory)
(defvoo nndir-get-new-mail nil nil nnml-get-new-mail nnmh-get-new-mail)

(defvoo nndir-status-string "" nil nnmh-status-string)
(defconst nndir-version "nndir 1.0")

;;; Interface functions.

(nnoo-define-basics nndir)

(deffoo nndir-open-server (server &optional defs)
  (setq nndir-directory
(or (cadr (assq 'nndir-directory defs))
    server))
  (unless (assq 'nndir-directory defs)
    (push `(nndir-directory ,server) defs))
  (push `(nndir-current-group
  ,(file-name-nondirectory (directory-file-name nndir-directory)))
defs)
  (push `(nndir-top-directory
  ,(file-name-directory (directory-file-name nndir-directory)))
defs)
  (nnoo-change-server 'nndir server defs))

(nnoo-map-functions nndir
  (nnml-retrieve-headers 0 nndir-current-group 0 0)
  (nnmh-request-article 0 nndir-current-group 0 0)
  (nnmh-request-group nndir-current-group 0 0)
```

```
  (nnmh-close-group nndir-current-group 0))

(nnoo-import nndir
  (nnmh
   nnmh-status-message
   nnmh-request-list
   nnmh-request-newgroups))

(provide 'nndir)
```

10.5.2 Score File Syntax

Score files are meant to be easily parsable, but yet extremely mallable. It was decided that something that had the same read syntax as an Emacs Lisp list would fit that spec.

Here's a typical score file:

```
(("summary"
  ("win95" -10000 nil s)
  ("Gnus"))
 ("from"
  ("Lars" -1000))
 (mark -100))
```

BNF definition of a score file:

```
score-file       = "" / "(" *element ")"
element          = rule / atom
rule             = string-rule / number-rule / date-rule
string-rule      = "(" quote string-header quote space *string-match ")"
number-rule      = "(" quote number-header quote space *number-match ")"
date-rule        = "(" quote date-header quote space *date-match ")"
quote            = <ascii 34>
string-header    = "subject" / "from" / "references" / "message-id" /
                   "xref" / "body" / "head" / "all" / "followup"
number-header    = "lines" / "chars"
date-header      = "date"
string-match     = "(" quote <string> quote [ "" / [ space score [ "" /
                   space date [ "" / [ space string-match-t ] ] ] ] ] ")"
score            = "nil" / <integer>
date             = "nil" / <natural number>
string-match-t   = "nil" / "s" / "substring" / "S" / "Substring" /
                   "r" / "regex" / "R" / "Regex" /
                   "e" / "exact" / "E" / "Exact" /
                   "f" / "fuzzy" / "F" / "Fuzzy"
number-match     = "(" <integer> [ "" / [ space score [ "" /
                   space date [ "" / [ space number-match-t ] ] ] ] ] ")"
number-match-t   = "nil" / "=" / "<" / ">" / ">=" / "<="
date-match       = "(" quote <string> quote [ "" / [ space score [ "" /
                   space date [ "" / [ space date-match-t ] ] ] ] ")"
date-match-t     = "nil" / "at" / "before" / "after"
```

```
atom             = "(" [ required-atom / optional-atom ] ")"
required-atom    = mark / expunge / mark-and-expunge / files /
                   exclude-files / read-only / touched
optional-atom    = adapt / local / eval
mark             = "mark" space nil-or-number
nil-or-number    = "nil" / <integer>
expunge          = "expunge" space nil-or-number
mark-and-expunge = "mark-and-expunge" space nil-or-number
files            = "files" *[ space <string> ]
exclude-files    = "exclude-files" *[ space <string> ]
read-only        = "read-only" [ space "nil" / space "t" ]
adapt            = "adapt" [ space "nil" / space "t" / space adapt-rule ]
adapt-rule       = "(" *[ <string> *[ "(" <string> <integer> ")" ] ")"
local            = "local" *[ space "(" <string> space <form> ")" ]
eval             = "eval" space <form>
space            = *[ " " / <TAB> / <NEWLINE> ]
```

Any unrecognized elements in a score file should be ignored, but not discarded.

As you can see, white space is needed, but the type and amount of white space is irrelevant. This means that formatting of the score file is left up to the programmer—if it's simpler to just spew it all out on one looong line, then that's ok.

The meaning of the various atoms are explained elsewhere in this manual.

10.5.3 Headers

Gnus uses internally a format for storing article headers that corresponds to the NOV format in a mysterious fashion. One could almost suspect that the author looked at the NOV specification and just shamelessly *stole* the entire thing, and one would be right.

Header is a severely overloaded term. "Header" is used in RFC1036 to talk about lines in the head of an article (eg., `From`). It is used by many people as a synonym for "head"—"the header and the body". (That should be avoided, in my opinion.) And Gnus uses a format internally that it calls "header", which is what I'm talking about here. This is a 9-element vector, basically, with each header (ouch) having one slot.

These slots are, in order: `number`, `subject`, `from`, `date`, `id`, `references`, `chars`, `lines`, `xref`. There are macros for accessing and setting these slots – they all have predictable names beginning with `mail-header-` and `mail-header-set-`, respectively.

The `xref` slot is really a `misc` slot. Any extra info will be put in there.

10.5.4 Ranges

GNUS introduced a concept that I found so useful that I've started using it a lot and have elaborated on it greatly.

The question is simple: If you have a large amount of objects that are identified by numbers (say, articles, to take a *wild* example) that you want to callify as being "included", a normal sequence isn't very useful. (A 200,000 length sequence is a bit long-winded.)

The solution is as simple as the question: You just collapse the sequence.

```
(1 2 3 4 5 6 10 11 12)
```

is transformed into

```
((1 . 6) (10 . 12))
```

To avoid having those nasty '`(13 . 13)`' elements to denote a lonesome object, a '`13`' is a valid element:

```
((1 . 6) 7 (10 . 12))
```

This means that comparing two ranges to find out whether they are equal is slightly tricky:

```
((1 . 5) 7 8 (10 . 12))
```

and

```
((1 . 5) (7 . 8) (10 . 12))
```

are equal. In fact, any non-descending list is a range:

```
(1 2 3 4 5)
```

is a perfectly valid range, although a pretty long-winded one. This is also legal:

```
(1 . 5)
```

and is equal to the previous range.

Here's a BNF definition of ranges. Of course, one must remember the semantic requirement that the numbers are non-descending. (Any number of repetition of the same number is allowed, but apt to disappear in range handling.)

```
range           = simple-range / normal-range
simple-range    = "(" number " . " number ")"
normal-range    = "(" start-contents ")"
contents        = "" / simple-range *[ " " contents ] /
                  number *[ " " contents ]
```

Gnus currently uses ranges to keep track of read articles and article marks. I plan on implementing a number of range operators in C if The Powers That Be are willing to let me. (I haven't asked yet, because I need to do some more thinking on what operators I need to make life totally range-based without ever having to convert back to normal sequences.)

10.5.5 Group Info

Gnus stores all permanent info on groups in a *group info* list. This list is from three to six elements (or more) long and exhaustively describes the group.

Here are two example group infos; one is a very simple group while the second is a more complex one:

```
("no.group" 5 (1 . 54324))

("nnml:my.mail" 3 ((1 . 5) 9 (20 . 55))
                  ((tick (15 . 19)) (replied 3 6 (19 . 3)))
                  (nnml "")
                  (auto-expire (to-address "ding@ifi.uio.no")))
```

The first element is the group name as Gnus knows the group; the second is the group level; the third is the read articles in range format; the fourth is a list of article marks lists; the fifth is the select method; and the sixth contains the group parameters.

Here's a BNF definition of the group info format:

```
info          = "(" group space level space read
                [ "" / [ space marks-list [ "" / [ space method [ "" /
                space parameters ] ] ] ] ] ")"
group         = quote <string> quote
level         = <integer in the range of 1 to inf>
read          = range
marks-lists   = nil / "(" *marks ")"
marks         = "(" <string> range ")"
method        = "(" <string> *elisp-forms ")"
parameters    = "(" *elisp-forms ")"
```

Actually that '`marks`' rule is a fib. A '`marks`' is a '`<string>`' consed on to a '`range`', but that's a bitch to say in pseudo-BNF.

10.5.6 Emacs/XEmacs Code

While Gnus runs under Emacs, XEmacs and Mule, I decided that one of the platforms must be the primary one. I chose Emacs. Not because I don't like XEmacs or Mule, but because it comes first alphabetically.

This means that Gnus will byte-compile under Emacs with nary a warning, while XEmacs will pump out gigabytes of warnings while byte-compiling. As I use byte-compilation warnings to help me root out trivial errors in Gnus, that's very useful.

I've also consistently used Emacs function interfaces, but have used Gnusey aliases for the functions. To take an example: Emacs defines a `run-at-time` function while XEmacs defines a `start-itimer` function. I then define a function called `gnus-run-at-time` that takes the same parameters as the Emacs `run-at-time`. When running Gnus under Emacs, the former function is just an alias for the latter. However, when running under XEmacs, the former is an alias for the following function:

```
(defun gnus-xmas-run-at-time (time repeat function &rest args)
  (start-itimer
   "gnus-run-at-time"
   `(lambda ()
      (,function ,@args))
   time repeat))
```

This sort of thing has been done for bunches of functions. Gnus does not redefine any native Emacs functions while running under XEmacs – it does this `defalias` thing with Gnus equivalents instead. Cleaner all over.

Of course, I could have chosen XEmacs as my native platform and done mapping functions the other way around. But I didn't. The performance hit these indirections impose on Gnus under XEmacs should be slight.

10.5.7 Various File Formats

10.5.7.1 Active File Format

The active file lists all groups that are available on the server in question. It also lists the highest and lowest current article numbers in each group.

Here's an excerpt from a typical active file:

```
soc.motss 296030 293865 y
alt.binaries.pictures.fractals 3922 3913 n
comp.sources.unix 1605 1593 m
comp.binaries.ibm.pc 5097 5089 y
no.general 1000 900 y
```

Here's a pseudo-BNF definition of this file:

```
active      = *group-line
group-line  = group space high-number space low-number space flag <NEWLINE>
group       = <non-white-space string>
space       = " "
high-number = <non-negative integer>
low-number  = <positive integer>
flag        = "y" / "n" / "m" / "j" / "x" / "=" group
```

10.5.7.2 Newsgroups File Format

The newsgroups file lists groups along with their descriptions. Not all groups on the server have to be listed, and not all groups in the file have to exist on the server. The file is meant purely as information to the user.

The format is quite simple; a group name, a tab, and the description. Here's the definition:

```
newsgroups    = *line
line          = group tab description <NEWLINE>
group         = <non-white-space string>
tab           = <TAB>
description   = <string>
```

10.6 Emacs for Heathens

Believe it or not, but some people who use Gnus haven't really used Emacs much before they embarked on their journey on the Gnus Love Boat. If you are one of those unfortunates whom "*M-C-a*", "kill the region", and "set `gnus-flargblossen` to an alist where the key is a regexp that is used for matching on the group name" are magical phrases with little or no meaning, then this appendix is for you. If you are already familiar with Emacs, just ignore this and go fondle your cat instead.

10.6.1 Keystrokes

- Q: What is an experienced Emacs user?
- A: A person who wishes that the terminal had pedals.

Yes, when you use Emacs, you are apt to use the control key, the shift key and the meta key a lot. This is very annoying to some people (notably `vi`le users), and the rest of us just love the hell out of it. Just give up and submit. Emacs really does stand for "Escape-Meta-Alt-Control-Shift", and not "Editing Macros", as you may have heard from other disreputable sources (like the Emacs author).

The shift key is normally located near your pinky fingers, and are normally used to get capital letters and stuff. You probably use it all the time. The control key is normally marked "CTRL" or something like that. The meta key is, funnily enough, never marked as such on any keyboards. The one I'm currently at has a key that's marked "Alt", which is the meta key on this keyboard. It's usually located somewhere to the left hand side of the keyboard, usually on the bottom row.

Now, us Emacs people doesn't say "press the meta-control-m key", because that's just too inconvenient. We say "press the *M-C-m* key". *M-* is the prefix that means "meta" and "C-" is the prefix that means "control". So "press *C-k*" means "press down the control key, and hold it down while you press *k*". "Press *M-C-k*" means "press down and hold down the meta key and the control key and then press *k*". Simple, ay?

This is somewhat complicated by the fact that not all keyboards have a meta key. In that case you can use the "escape" key. Then *M-k* means "press escape, release escape, press *k*". That's much more work than if you have a meta key, so if that's the case, I respectfully suggest you get a real keyboard with a meta key. You can't live without it.

10.6.2 Emacs Lisp

Emacs is the King of Editors because it's really a Lisp interpreter. Each and every key you tap runs some Emacs Lisp code snippet, and since Emacs Lisp is an interpreted language, that means that you can configure any key to run any arbitrary code. You just, like, do it.

Gnus is written in Emacs Lisp, and is run as a bunch of interpreted functions. (These are byte-compiled for speed, but it's still interpreted.) If you decide that you don't like the way Gnus does certain things, it's trivial to have it do something a different way. (Well, at least if you know how to write Lisp code.) However, that's beyond the scope of this manual, so we are simply going to talk about some common constructs that you normally use in your '`.emacs`' file to customize Gnus.

If you want to set the variable `gnus-florgbnize` to four (4), you write the following:

```
(setq gnus-florgbnize 4)
```

This function (really "special form") `setq` is the one that can set a variable to some value. This is really all you need to know. Now you can go and fill your `.emacs` file with lots of these to change how Gnus works.

If you have put that thing in your `.emacs` file, it will be read and `eval`ed (which is lisp-ese for "run") the next time you start Emacs. If you want to change the variable right away, simply say `C-x C-e` after the closing parenthesis. That will `eval` the previous "form", which here is a simple `setq` statement.

Go ahead—just try it, if you're located at your Emacs. After you `C-x C-e`, you will see '`4`' appear in the echo area, which is the return value of the form you `eval`ed.

Some pitfalls:

If the manual says "set `gnus-read-active-file` to `some`", that means:

```
(setq gnus-read-active-file 'some)
```

On the other hand, if the manual says "set `gnus-nntp-server` to '`nntp.ifi.uio.no`'", that means:

```
(setq gnus-nntp-server "nntp.ifi.uio.no")
```

So be careful not to mix up strings (the latter) with symbols (the former). The manual is unambiguous, but it can be confusing.

\input texinfo

10.7 Frequently Asked Questions

This is the Gnus Frequently Asked Questions list. If you have a Web browser, the official hypertext version is at '`http://www.miranova.com/~steve/gnus-faq.html>`', and has probably been updated since you got this manual.

10.7.1 Installation

- Q1.1 What is the latest version of Gnus?

  The latest (and greatest) version is 5.0.10. You might also run across something called *September Gnus*. September Gnus is the alpha version of the next major release of Gnus. It is currently not stable enough to run unless you are prepared to debug lisp.

- Q1.2 Where do I get Gnus?

  Any of the following locations:

  - '`ftp://ftp.ifi.uio.no/pub/emacs/gnus/gnus.tar.gz`'
  - '`ftp://ftp.pilgrim.umass.edu/pub/misc/ding/`'
  - '`gopher://gopher.pilgrim.umass.edu/11/pub/misc/ding/`'
  - '`ftp://aphrodite.nectar.cs.cmu.edu/pub/ding-gnus/`'
  - '`ftp://ftp.solace.mh.se:/pub/gnu/elisp/`'

- Q1.3 Which version of Emacs do I need?

  At least GNU Emacs 19.28, or XEmacs 19.12 is recommended. GNU Emacs 19.25 has been reported to work under certain circumstances, but it doesn't *officially* work on it. 19.27 has also been reported to work. Gnus has been reported to work under OS/2 as well as Unix.

- Q1.4 Where is timezone.el?

  Upgrade to XEmacs 19.13. In earlier versions of XEmacs this file was placed with Gnus 4.1.3, but that has been corrected.

- Q1.5 When I run Gnus on XEmacs 19.13 I get weird error messages.

  You're running an old version of Gnus. Upgrade to at least version 5.0.4.

- Q1.6 How do I unsubscribe from the Mailing List?

  Send an e-mail message to '`ding-request@ifi.uio.no`' with the magic word *unsubscribe* somewhere in it, and you will be removed.

  If you are reading the digest version of the list, send an e-mail message to '`ding-rn-digests-d-request@moe.shore.net`' with *unsubscribe* as the subject and you will be removed.

- Q1.7 How do I run Gnus on both Emacs and XEmacs?

  The basic answer is to byte-compile under XEmacs, and then you can run under either Emacsen. There is, however, a potential version problem with easymenu.el with Gnu Emacs prior to 19.29.

  Per Abrahamsen <abraham@dina.kvl.dk> writes :
  The internal easymenu.el interface changed between 19.28 and 19.29 in order to make it possible to create byte compiled files that can be shared between Gnu Emacs and XEmacs. The change is upward compatible, but not downward compatible. This gives the following compatibility table:

```
Compiled with:  | Can be used with:
----------------+-------------------------------------
19.28           | 19.28         19.29
19.29           |               19.29           XEmacs
XEmacs          |               19.29           XEmacs
```

  If you have Gnu Emacs 19.28 or earlier, or XEmacs 19.12 or earlier, get a recent version of auc-menu.el from '`ftp://ftp.iesd.auc.dk/pub/emacs-lisp/auc-menu.el`', and install it under the name easymenu.el somewhere early in your load path.

- Q1.8 What resources are available?

  There is the newsgroup Gnu.emacs.gnus. Discussion of Gnus 5.x is now taking place there. There is also a mailing list, send mail to '`ding-request@ifi.uio.no`' with the magic word *subscribe* somewhere in it.

  NOTE: the traffic on this list is heavy so you may not want to be on it (unless you use Gnus as your mailer reader, that is). The mailing list is mainly for developers and testers.

  Gnus has a home World Wide Web page at
  '`http://www.ifi.uio.no/~larsi/ding.html`'.

  Gnus has a write up in the X Windows Applications FAQ at
  '`http://www.ee.ryerson.ca:8080/~elf/xapps/Q-III.html`'.

  The Gnus manual is also available on the World Wide Web. The canonical source is in Norway at
  '`http://www.ifi.uio.no/~larsi/ding-manual/gnus_toc.html`'.

  There are three mirrors in the United States:

1. 'http://www.miranova.com/gnus-man/'
2. 'http://www.pilgrim.umass.edu/pub/misc/ding/manual/gnus_toc.html'
3. 'http://www.rtd.com/~woo/gnus/'

PostScript copies of the Gnus Reference card are available from 'ftp://ftp.cs.ualberta.ca/pub/oolog/gnus/'. They are mirrored at 'ftp://ftp.pilgrim.umass.edu/pub/misc/ding/refcard/' in the United States. And 'ftp://marvin.fkphy.uni-duesseldorf.de/pub/gnus/' in Germany.

An online version of the Gnus FAQ is available at 'http://www.miranova.com/~steve/gnus-faq.html'. Off-line formats are also available:
ASCII: 'ftp://ftp.miranova.com/pub/gnus/gnus-faq'
PostScript: 'ftp://ftp.miranova.com/pub/gnus/gnus-faq.ps'.

- Q1.9 Gnus hangs on connecting to NNTP server

  I am running XEmacs on SunOS and Gnus prints a message about Connecting to NNTP server and then just hangs.

  Ben Wing <wing@netcom.com> writes :
  I wonder if you're hitting the infamous *libresolv* problem. The basic problem is that under SunOS you can compile either with DNS or NIS name lookup libraries but not both. Try substituting the IP address and see if that works; if so, you need to download the sources and recompile.

- Q1.10 Mailcrypt 3.4 doesn't work

  This problem is verified to still exist in Gnus 5.0.9 and MailCrypt 3.4. The answer comes from Peter Arius <arius@immd2.informatik.uni-erlangen.de>.

  I found out that mailcrypt uses `gnus-eval-in-buffer-window`, which is a macro. It seems as if you have compiled mailcrypt with plain old GNUS in load path, and the XEmacs byte compiler has inserted that macro definition into 'mc-toplev.elc'. The solution is to recompile 'mc-toplev.el' with Gnus 5 in load-path, and it works fine.

  Steve Baur <steve@miranova.com> adds :
  The problem also manifests itself if neither GNUS 4 nor Gnus 5 is in the load-path.

- Q1.11 What other packages work with Gnus?
  - Mailcrypt.

    Mailcrypt is an Emacs interface to PGP. It works, it installs without hassle, and integrates very easily. Mailcrypt can be obtained from 'ftp://cag.lcs.mit.edu/pub/patl/mailcrypt-3.4.tar.gz'.

  - Tiny Mime.

    Tiny Mime is an Emacs MUA interface to MIME. Installation is a two-step process unlike most other packages, so you should be prepared to move the byte-compiled code somewhere. There are currently two versions of this package available. It can be obtained from 'ftp://ftp.jaist.ac.jp/pub/GNU/elisp/'. Be sure to apply the supplied patch. It works with Gnus through version 5.0.9. In order for all dependencies to work correctly the load sequence is as follows:

```
(load "tm-setup")
(load "gnus")
(load "mime-compose")
```

NOTE: Loading the package disables citation highlighting by default. To get the old behavior back, use the *M-t* command.

10.7.2 Customization

- Q2.1 Custom Edit does not work under XEmacs

  The custom package has not been ported to XEmacs.

- Q2.2 How do I quote messages?

  I see lots of messages with quoted material in them. I am wondering how to have Gnus do it for me.

  This is Gnus, so there are a number of ways of doing this. You can use the built-in commands to do this. There are the *F* and *R* keys from the summary buffer which automatically include the article being responded to. These commands are also selectable as *Followup and Yank* and *Reply and Yank* in the Post menu.

  C-c C-y grabs the previous message and prefixes each line with `ail-indentation-spaces` spaces or `mail-yank-prefix` if that is non-nil, unless you have set your own `mail-citation-hook`, which will be called to to do the job.

  You might also consider the Supercite package, which allows for pretty arbitrarily complex quoting styles. Some people love it, some people hate it.

- Q2.3 How can I keep my nnvirtual:* groups sorted?

  How can I most efficiently arrange matters so as to keep my nnvirtual:* (etc) groups at the top of my group selection buffer, whilst keeping everything sorted in alphabetical order.

  If you don't subscribe often to new groups then the easiest way is to first sort the groups and then manually kill and yank the virtuals wherever you want them.

- Q2.4 Any good suggestions on stuff for an all.SCORE file?

  Here is a collection of suggestions from the Gnus mailing list.

  1. From "Dave Disser" <disser@sdd.hp.com>
     I like blasting anything without lowercase letters. Weeds out most of the make $$ fast, as well as the lame titles like "IBM" and "HP-UX" with no further description.

     ```
     (("Subject"
      ("^\\(Re: \\)?[^a-z]*$" -200 nil R)))
     ```

  2. From "Peter Arius" <arius@immd2.informatik.uni-erlangen.de>
     The most vital entries in my (still young) all.SCORE:

     ```
     (("xref"
       ("alt.fan.oj-simpson" -1000 nil s))
      ("subject"
       ("\\<\\(make\\|fast\\|big\\)\\s-*\\(money\\|cash\\|bucks?\\)\\>" -1000 nil r)
       ("$$$$" -1000 nil s)))
     ```

3. From “Per Abrahamsen” <abraham@dina.kvl.dk>

```
(("subject"
  ;; CAPS OF THE WORLD, UNITE
  ("^..[^a-z]+$" -1 nil R)
  ;; $$$ Make Money $$$ (Try work)
  ("$" -1 nil s)
  ;; I'm important! And I have exclamation marks to prove it!
  ("!" -1 nil s)))
```

4. From “heddy boubaker” <boubaker@cenatls.cena.dgac.fr>
   I would like to contribute with mine.

```
(
 (read-only t)
 ("subject"
  ;; ALL CAPS SUBJECTS
  ("^\\([Rr][Ee]: +\\)?[^a-z]+$" -1 nil R)
  ;; $$$ Make Money $$$
  ("$$" -10 nil s)
  ;; Empty subjects are worthless!
  ("^ *\\([[(<]none[>)]\\|(no subject\\( given\\)?)\\)? *$" -10 nil r)
  ;; Sometimes interesting announces occur!
  ("ANN?OU?NC\\(E\\|ING\\)" +10 nil r)
  ;; Some people think they're on mailing lists
  ("\\(un\\)?sub?scribe" -100 nil r)
  ;; Stop Micro$oft NOW!!
  ("\\(m\\(icro\\)?[s$]\\(oft\\|lot\\)?-?\\)?wind?\\(ows\\|aube\\|oze\\)?[- ]*\
  ;; I've nothing to buy
  ("\\(for\\|4\\)[- ]*sale" -100 nil r)
  ;; SELF-DISCIPLINED people
  ("\\[[^a-z0-9 \t\n][^a-z0-9 \t\n]\\]" +100 nil r)
  )
 ("from"
  ;; To keep track of posters from my site
  (".dgac.fr" +1000 nil s))
 ("followup"
  ;; Keep track of answers to my posts
  ("boubaker" +1000 nil s))
 ("lines"
  ;; Some people have really nothing to say!!
  (1 -10 nil <=))
 (mark -100)
 (expunge -1000)
 )
```

5. From “Christopher Jones” <cjones@au.oracle.com>
   The sample ‘`all.SCORE`’ files from Per and boubaker could be augmented with:

```
(("subject"
  ;; No junk mail please!
```

```
        ("please ignore" -500 nil s)
        ("test" -500 nil e))
      )
```

6. From “Brian Edmonds” <edmonds@cs.ubc.ca>
   Augment any of the above with a fast method of scoring down excessively cross posted articles.

```
("xref"
 ;; the more cross posting, the exponentially worse the article
 ("^xref: \\S-+ \\S-+ \\S-+ \\S-+" -1 nil r)
 ("^xref: \\S-+ \\S-+ \\S-+ \\S-+ \\S-+" -2 nil r)
 ("^xref: \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+" -4 nil r)
 ("^xref: \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+" -8 nil r)
 ("^xref: \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+" -16 nil r)
 ("^xref: \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+" -32 nil r)
 ("^xref: \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+" -64 nil
 ("^xref: \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+" -
 ("^xref: \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\
 ("^xref: \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\S-+ \\
```

- Q2.5 What do I use to yank-through when replying?

  You should probably reply and followup with `R` and `F`, instead of `r` and `f`, which solves your problem. But you could try something like:

```
(defconst mail-yank-ignored-headers
  "^.*:"
  "Delete these headers from old message when it's inserted in a reply.")
```

- Q2.6 I don't like the default WWW browser

  Now when choosing an URL Gnus starts up a W3 buffer, I would like it to always use Netscape (I don't browse in text-mode ;-).

  1. Activate ‘Customize...’ from the ‘Help’ menu.
  2. Scroll down to the ‘WWW Browser’ field.
  3. Click ‘mouse-2’ on ‘WWW Browser’.
  4. Select ‘Netscape’ from the pop up menu.
  5. Press ‘C-c C-c’

  If you are using XEmacs then to specify Netscape do

```
(setq gnus-button-url 'gnus-netscape-open-url)
```

- Q2.7 What, if any, relation is between “ask-server” and “(setq gnus-read-active-file 'some)”?

  In order for Gnus to show you the complete list of newsgroups, it will either have to either store the list locally, or ask the server to transmit the list. You enable the first with

```
(setq gnus-save-killed-list t)
```

  and the second with

```
(setq gnus-read-active-file t)
```

  If both are disabled, Gnus will not know what newsgroups exists. There is no option to get the list by casting a spell.

- Q2.8 Moving between groups is slow.

  Per Abrahamsen <abraham@dina.kvl.dk> writes:

  Do you call `define-key` or something like that in one of the summary mode hooks? This would force Emacs to recalculate the keyboard shortcuts. Removing the call should speed up *M-x gnus-summary-mode RET* by a couple of orders of magnitude. You can use

  ```
  (define-key gnus-summary-mode-map KEY COMMAND)
  ```

  in your '`.gnus`' instead.

10.7.3 Reading News

- Q3.1 How do I convert my kill files to score files?

  A kill-to-score translator was written by Ethan Bradford <ethanb@ptolemy.astro.washington.edu>. It is available from '`http://baugi.ifi.uio.no/~larsi/ding-various/gnus-kill-to-score.el`'.

- Q3.2 My news server has a lot of groups, and killing groups is painfully slow.

  Don't do that then. The best way to get rid of groups that should be dead is to edit your newsrc directly. This problem will be addressed in the near future.

- Q3.3 How do I use an NNTP server with authentication?

  Put the following into your .gnus:

  ```
  (add-hook 'nntp-server-opened-hook 'nntp-send-authinfo)
  ```

- Q3.4 Not reading the first article.

  How do I avoid reading the first article when a group is selected?

  1. Use *RET* to select the group instead of *SPC*.
  2. `(setq gnus-auto-select first nil)`
  3. Luis Fernandes <elf@mailhost.ee.ryerson.ca>writes:
     This is what I use...customize as necessary...

     ```
     ;;; Don't auto-select first article if reading sources, or archives or
     ;;; jobs postings, etc. and just display the summary buffer
     (add-hook 'gnus-select-group-hook
       (function
        (lambda ()
          (cond ((string-match "sources" gnus-newsgroup-name)
         (setq gnus-auto-select-first nil))
        ((string-match "jobs" gnus-newsgroup-name)
     (setq gnus-auto-select-first nil))
        ((string-match "comp\\.archives" gnus-newsgroup-name)
         (setq gnus-auto-select-first nil))
        ((string-match "reviews" gnus-newsgroup-name)
         (setq gnus-auto-select-first nil))
        ((string-match "announce" gnus-newsgroup-name)
         (setq gnus-auto-select-first nil))
     ```

```
((string-match "binaries" gnus-newsgroup-name)
 (setq gnus-auto-select-first nil))
(t
 (setq gnus-auto-select-first t))))))
```

4. Per Abrahamsen <abraham@dina.kvl.dk> writes:
   Another possibility is to create an '`all.binaries.all.SCORE`' file like this:

   ```
   ((local
     (gnus-auto-select-first nil)))
   ```

   and insert

   ```
   (setq gnus-auto-select-first t)
   ```

   in your '`.gnus`'.

- Q3.5 Why aren't BBDB known posters marked in the summary buffer?

  Brian Edmonds <edmonds@cs.ubc.ca> writes:
  Due to changes in Gnus 5.0, '`bbdb-gnus.el`' no longer marks known posters in the summary buffer. An updated version, '`gnus-bbdb.el`' is available at the locations listed below. This package also supports autofiling of incoming mail to folders specified in the BBDB. Extensive instructions are included as comments in the file.

  Send mail to '`majordomo@edmonds.home.cs.ubc.ca`' with the following line in the body of the message: *get misc gnus-bbdb.el*.

  Or get it from the World Wide Web:
  '`http://www.cs.ubc.ca/spider/edmonds/gnus-bbdb.el`'.

10.7.4 Reading Mail

- Q4.1 What does the message "Buffer has changed on disk" mean in a mail group?

  Your filter program should not deliver mail directly to your folders, instead it should put the mail into spool files. Gnus will then move the mail safely from the spool files into the folders. This will eliminate the problem. Look it up in the manual, in the section entitled "Mail & Procmail".

- Q4.2 How do you make articles un-expirable?

  I am using nnml to read news and have used `gnus-auto-expirable-newsgroups` to automagically expire articles in some groups (Gnus being one of them). Sometimes there are interesting articles in these groups that I want to keep. Is there any way of explicitly marking an article as un-expirable - that is mark it as read but not expirable?

  Use *u*, *!*, *d* or *M-u* in the summary buffer. You just remove the *E* mark by setting some other mark. It's not necessary to tick the articles.

- Q4.3 How do I delete bogus nnml: groups?

  My problem is that I have various mail (nnml) groups generated while experimenting with Gnus. How do I remove them now? Setting the level to 9 does not help. Also `gnus-group-check-bogus-groups` does not recognize them.

  Removing mail groups is tricky at the moment. (It's on the to-do list, though.) You basically have to kill the groups in Gnus, shut down Gnus, edit the active file to exclude

these groups, and probably remove the nnml directories that contained these groups as well. Then start Gnus back up again.

- Q4.4 What happened to my new mail groups?

  I got new mail, but I have never seen the groups they should have been placed in.

  They are probably there, but as zombies. Press `A z` to list zombie groups, and then subscribe to the groups you want with `u`. This is all documented quite nicely in the user's manual.

- Q4.5 Not scoring mail groups

  How do you *totally* turn off scoring in mail groups?

  Use an nnbabyl:all.SCORE (or nnmh, or nnml, or whatever) file containing:

```
((adapt ignore)
 (local (gnus-use-scoring nil))
 (exclude-files "all.SCORE"))
```

11 Index

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z

12 Key Index

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

Short Contents

Table of Contents