A Guide to the Mazes of Menace:
Guidebook for *NetHack*

Original version - Eric S. Raymond
(Edited and expanded for 5.0.0 by Mike Stephenson and others)

May 2, 2026

1 Introduction

Recently, you have begun to find yourself unfulfilled and distant in your daily occupation. Strange dreams of prospecting, stealing, crusading, and combat have haunted you in your sleep for many months, but you aren't sure of the reason. You wonder whether you have in fact been having those dreams all your life, and somehow managed to forget about them until now. Some nights you awaken suddenly and cry out, terrified at the vivid recollection of the strange and powerful creatures that seem to be lurking behind every corner of the dungeon in your dream. Could these details haunting your dreams be real? As each night passes, you feel the desire to enter the mysterious caverns near the ruins grow stronger. Each morning, however, you quickly put the idea out of your head as you recall the tales of those who entered the caverns before you and did not return. Eventually you can resist the yearning to seek out the fantastic place in your dreams no longer. After all, when other adventurers came back this way after spending time in the caverns, they usually seemed better off than when they passed through the first time. And who was to say that all of those who did not return had not just kept going?

Asking around, you hear about a bauble, called the Amulet of Yendor by some, which, if you can find it, will bring you great wealth. One legend you were told even mentioned that the one who finds the amulet will be granted immortality by the gods. The amulet is rumored to be somewhere beyond the Valley of Gehennom, deep within the Mazes of Menace. Upon hearing the legends, you immediately realize that there is some profound and undiscovered reason that you are to descend into the caverns and seek out that amulet of which they spoke. Even if the rumors of the amulet's powers are untrue, you decide that you should at least be able to sell the tales of your adventures to the local minstrels for a tidy sum, especially if you encounter any of the terrifying and magical creatures of your dreams along the way. You spend one last night fortifying yourself at the local inn, becoming more and more depressed as you watch the odds of your success being posted on the inn's walls getting lower and lower.
In the morning you awake, collect your belongings, and set off for the dungeon. After several days of uneventful travel, you see the ancient ruins that mark the entrance to the Mazes of Menace. It is late at night, so you make camp at the entrance and spend the night sleeping under the open skies. In the morning, you gather your gear, eat what may be your last meal outside, and enter the dungeon...

2 What is going on here?

You have just begun a game of *NetHack*. Your goal is to grab as much treasure as you can, retrieve the Amulet of Yendor, and escape the Mazes of Menace alive.

Your abilities and strengths for dealing with the hazards of adventure will vary with your background and training:

Archeologists understand dungeons pretty well; this enables them to move quickly and sneak up on the local nasties. They start equipped with the tools for a proper scientific expedition, and are able to read ancient languages.

Barbarians are warriors out of the hinterland, hardened to battle. They begin their quests with naught but uncommon strength, a trusty hauberk, and a great two-handed sword.

Cavemen and **Cavewomen** start with exceptional strength, but unfortunately, neolithic weapons.

Healers are wise in medicine and apothecary. They know the herbs and simples that can restore vitality, ease pain, anesthetize, and neutralize poisons; and with their instruments, they can divine a being's state of health or sickness. Their medical practice earns them quite reasonable amounts of money, with which they enter the dungeon.

Knights are distinguished from the common skirmisher by their devotion to the ideals of chivalry and by the surpassing excellence of their armor.

Monks are ascetics, who by rigorous practice of physical and mental disciplines have become capable of fighting as effectively without weapons as with. They wear no armor but make up for it with increased mobility.

Priests and **Priestesses** are clerics militant, crusaders advancing the cause of righteousness with arms, armor, and arts thaumaturgic. Their ability to commune with deities via prayer occasionally extricates them from peril, but can also put them in it.

Rangers are most at home in the woods, and some say slightly out of place in a dungeon. They are, however, experts in archery as well as tracking and stealthy movement.

Rogues are agile and stealthy thieves, with knowledge of locks, traps, and poisons. Their advantage lies in surprise, which they employ to great advantage.

Samurai are the elite warriors of feudal Nippon. They are lightly armored and quick, and wear the *dai-sho*, two swords of the deadliest keenness.

Tourists start out with lots of gold (suitable for shopping with), a credit card, lots of food, some maps, and an expensive camera. Most monsters don't like being photographed.

Valkyries are hardy warrior women. Their upbringing in the harsh Northlands makes them strong, inures them to extremes of cold, and instills in them stealth and cunning.

Wizards start out with a knowledge of magic, a selection of magical items, and a particular affinity for dweomercraft. Although seemingly weak and easy to overcome at first sight, an experienced Wizard is a deadly foe.

You may also choose the race of your character (within limits; most roles have restrictions on which races are eligible for them):

Dwarves are smaller than humans or elves, but are stocky and solid individuals. Dwarves' most notable trait is their great expertise in mining and metalwork. Dwarvish armor is said to be second in quality not even to the mithril armor of the Elves.

Elves are agile, quick, and perceptive; very little of what goes on will escape an Elf. The quality of Elven craftsmanship often gives them an advantage in arms and armor.

Gnomes are smaller than but generally similar to dwarves. Gnomes are known to be expert miners, and it is known that a secret underground mine complex built by this race exists within the Mazes of Menace, filled with both riches and danger.

Humans are by far the most common race of the surface world, and are thus the norm to which other races are often compared. Although they have no special abilities, they can succeed in any role.

Orcs are a cruel and barbaric race that hate every living thing (including other orcs). Above all others, Orcs hate Elves with a passion unequalled, and will go out of their way to kill one at any opportunity. The armor and weapons fashioned by the Orcs are typically of inferior quality.

3 What do all those things on the screen mean?

On the screen is kept a map of where you have been and what you have seen on the current dungeon level; as you explore more of the level, it appears on the screen in front of you.

When *NetHack*'s ancestor *rogue* first appeared, its screen orientation was almost unique among computer fantasy games. Since then, screen orientation has become the norm rather than the exception; *NetHack* continues this fine tradition. Unlike text adventure games that accept commands in pseudo-English sentences and explain the results in words, *NetHack* commands are all one or two keystrokes and the results are displayed graphically on the screen. A minimum screen size of 24 lines by 80 columns is recommended; if the screen is larger, only a 21×80 section will be used for the map.

NetHack can even be played by blind players, with the assistance of Braille readers or speech synthesisers. Instructions for configuring *NetHack* for the blind are included later in this document.

NetHack generates a new dungeon every time you play it; even the authors still find it an entertaining and exciting game despite having won several times.

NetHack offers a variety of display options. The options available to you will vary from port to port, depending on the capabilities of your hardware and software, and whether various compile-time options were enabled when your executable was created. The three possible display options are: a monochrome character interface, a color character interface, and a graphical interface using small pictures called tiles. The two character interfaces allow fonts with other characters to be substituted, but the default assignments use standard ASCII characters to represent everything. There is no difference between the various display options with respect to game play. Because we cannot reproduce the tiles or colors in the Guidebook, and because it is common to all ports, we will use the default ASCII characters from the monochrome character display when referring to things you might see on the screen during your game. In order to understand what is going on in *NetHack*, first you must understand what *NetHack* is doing with the screen. The *NetHack* screen replaces the "You see ..." descriptions of text adventure games. Figure 1 is a sample of what a *NetHack* screen might look like. The way the screen looks for you depends on your platform.

```
The bat bites!


        ------
        |....|    ----------
        |.<..|####...@...$.|
        |....-#   |...B....+
        |....|    |.d......|
        ------    -------|--



Player the Rambler   St:12 Dx:7 Co:18 In:11 Wi:9 Ch:15 Neutral
Dlvl:1 $:993 HP:9(12) Pw:3(3) AC:10 Exp:1/19 T:752 Hungry Conf
```

Figure 1

```
Player the Rambler   St:12 Dx:7 Co:18 In:11 Wi:9 Ch:15
Neutral $:993 HP:9(12) Pw:3(3) AC:10 Exp:1/19 Hungry
Dlvl:1 T:752                                  Conf
```

Figure 2

The status lines (bottom)

The bottom two (or three) lines of the screen contain several cryptic pieces of information describing your current status. Figure 1 shows the traditional two-line status area below the map. Figure 2 shows just the status area, when the *statuslines:3* option has been set (not all interfaces support this option). If any status line becomes wider than the screen, you might not see all of it due to truncation. When the numbers grow bigger and multiple *conditions* are present, the two-line format will run out of room on the second line, but *statuslines:2* is the default because a basic 24-line terminal isn't tall enough for the third line.

Here are explanations of what the various status items mean:

Title Your character's name and professional ranking (based on role and *experience level*, see below).

Strength A measure of your character's strength; one of your six basic attributes. A human character's attributes can range from 3 to 18 inclusive; non-humans may exceed these limits (occasionally you may get super-strengths of the form 18/xx, and magic can

also cause attributes to exceed the normal limits). The higher your strength, the stronger you are. Strength affects how successfully you perform physical tasks, how much damage you do in combat, and how much loot you can carry.

Dexterity Dexterity affects your chances to hit in combat, to avoid traps, and do other tasks requiring agility or manipulation of objects.

Constitution Constitution affects your ability to recover from injuries and other strains on your stamina. When strength is low or modest, constitution also affects how much you can carry. With sufficiently high strength, the contribution to carrying capacity from your constitution no longer matters.

Intelligence Intelligence affects your ability to cast spells and read spellbooks.

Wisdom Wisdom comes from your practical experience (especially when dealing with magic). It affects your magical energy.

Charisma Charisma affects how certain creatures react toward you. In particular, it can affect the prices shopkeepers offer you.

Alignment *Lawful*, *Neutral* or *Chaotic*. Often, Lawful is taken as good and Chaotic as evil, but legal and ethical do not always coincide. Your alignment influences how other monsters react toward you. Monsters of a like alignment are more likely to be non-aggressive, while those of an opposing alignment are more likely to be seriously offended at your presence.

Dungeon Level How deep you are in the dungeon. You start at level one and the number increases as you go deeper into the dungeon. Some levels are special, and are identified by a name and not a number. The Amulet of Yendor is reputed to be somewhere beneath the twentieth level.

Gold The number of gold pieces you are openly carrying. Gold which you have concealed in containers is not counted.

Hit Points Your current and maximum hit points. Hit points indicate how much damage you can take before you die. The more you get hit in a fight, the lower they get. You can regain hit points by resting, or by using certain magical items or spells. The number in parentheses is the maximum number your hit points can reach.

Power Spell points. This tells you how much mystic energy (*mana*) you have available for spell casting. Again, resting will regenerate the amount available.

Armor Class A measure of how effectively your armor stops blows from unfriendly creatures. The lower this number is, the more effective the armor; it is quite possible to have negative armor class. See the *Armor* subsection of *Objects* for more information.

Experience Your current experience level. If the *showexp* option is set, it will be followed by a slash and experience points. As you adventure, you gain experience points. At certain experience point totals, you gain an experience level. The more experienced you are, the better you fight and withstand magical attacks. (By the time your level reaches double digits, the usefulness of showing the points with it has dropped significantly. You can use the '`O`' command to turn *showexp* off to avoid using up the limited status line space.)

Time The number of turns elapsed so far, displayed if you have the *time* option set.

Status Hunger: your current hunger status. Values are *Satiated*, *Not Hungry* (or *Normal*), *Hungry*, *Weak*, and *Fainting*. Not shown when *Normal*.
Encumbrance: an indication of how what you are carrying affects your ability to move. Values are *Unencumbered*, *Burdened*, *Stressed*, *Strained*, *Overtaxed*, and *Overloaded*. Not shown when *Unencumbered*.
Fatal conditions: *Stone* (aka *Petrifying*, turning to stone), *Slime* (turning into green slime), *Strngl* (being strangled), *FoodPois* (suffering from acute food poisoning), *Ter-*

mIll (suffering from a terminal illness).
Non-fatal conditions: *Blind* (can't see), *Deaf* (can't hear), *Stun* (stunned), *Conf* (confused), *Hallu* (hallucinating).
Movement modifiers: *Lev* (levitating), *Fly* (flying), *Ride* (riding).
Other conditions and modifiers exist, but there isn't enough room to display them with the other status fields.
The `#attributes` command (default key `^X`) will show all current status information in unabbreviated format. It also shows other information which might be included on the status lines if those had more room.

The message line (top)

The top line of the screen is reserved for messages that describe things that are impossible to represent visually. If you see a "`--More--`" on the top line, this means that *NetHack* has another message to display on the screen, but it wants to make certain that you've read the one that is there first. To read the next message, just press the space bar.

To change how and what messages are shown on the message line, see "*Configuring Message Types*" and the *verbose* option.

The map (rest of the screen)

The rest of the screen is the map of the level as you have explored it so far. Each symbol on the screen represents something. You can set various graphics options to change some of the symbols the game uses; otherwise, the game will use default symbols. Here is a list of what the default symbols mean:

`-`	The horizontal or corner walls of a room, or an open east/west door.
`|`	The vertical walls of a room, or an open north/south door, or a grave.
`.`	The floor of a room, or ice, or a doorless doorway, or the span of an open drawbridge.
`#`	A corridor, or iron bars, or a tree, or the portcullis of a closed drawbridge. Note: engravings in corridors also appear as # but are shown in a different color from normal corridor locations.
`>`	Stairs down: a way to the next level.
`<`	Stairs up: a way to the previous level.
`+`	A closed door, or a spellbook containing a spell you may be able to learn.
`@`	Your character or a human or an elf.
`$`	A pile of gold.
`^`	A trap (once you have detected it).
`)`	A weapon.
`[`	A suit or piece of armor.
`%`	Something edible (not necessarily healthy).
`?`	A scroll.
`/`	A wand.
`=`	A ring.
`!`	A potion.
`(`	A useful item (pick-axe, key, lamp ...).
`"`	An amulet or a spider web.
`*`	A gem or rock (possibly valuable, possibly worthless).

`` ` ``	A boulder or statue or an engraving on the floor of a room. Note: statues are displayed as if they were the monsters they depict so won't appear as a *grave accent* (aka *back-tick*).
`0`	An iron ball.
`_`	An altar, or an iron chain.
{	A fountain or a sink.
}	A pool of water or moat or a wall of water or a pool of lava or a wall of lava.
\	An opulent throne.
`a-z` and `A-HJ-Z` and `@&':;`	Letters and certain other symbols represent the various inhabitants of the Mazes of Menace. Watch out, they can be nasty and vicious. Sometimes, however, they can be helpful.
`I`	Rather than a specific type of monster, this marks the last known location of an invisible or otherwise unseen monster. Note that the monster could have moved. The '`s`', '`F`', and '`m`' commands may be useful here.
`1-5`	The digits 1 through 5 may be displayed, marking unseen monsters sensed via the *Warning* attribute. Less dangerous monsters are indicated by lower values, more dangerous by higher values.

You need not memorize all these symbols; you can ask the game what any symbol represents with the '`/`' command (see the next section for more info).

4 Commands

Commands can be initiated by typing one or two characters to which the command is bound to, or typing the command name in the extended commands entry. Some commands, like "`search`", do not require that any more information be collected by *NetHack*. Other commands might require additional information, for example a direction, or an object to be used. For those commands that require additional information, *NetHack* will present you with either a menu of choices, or with a command line prompt requesting information. Which you are presented with will depend chiefly on how you have set the '*menustyle*' option.

For example, a common question in the form "`What do you want to use? [a-zA-Z ?*]`", asks you to choose an object you are carrying. Here, "`a-zA-Z`" are the inventory letters of your possible choices. Typing '`?`' gives you an inventory list of these items, so you can see what each letter refers to. In this example, there is also a '`*`' indicating that you may choose an object not on the list, if you wanted to use something unexpected. Typing a '`*`' lists your entire inventory, so you can see the inventory letters of every object you're carrying. Finally, if you change your mind and decide you don't want to do this command after all, you can press the 'ESC' key to abort the command.

You can put a number before some commands to repeat them that many times; for example, "`10s`" will search ten times. If you have the *number_pad* option set, you must type '`n`' to prefix a count, so the example above would be typed "`n10s`" instead. Commands for which counts make no sense ignore them. In addition, movement commands can be prefixed for greater control (see below). To cancel a count or a prefix, press the 'ESC' key.

The list of commands is rather long, but it can be read at any time during the game through the '`?`' command, which accesses a menu of helpful texts. Here are the default key bindings for your reference:

`?`	Help menu: display one of several help texts available.
`/`	The `whatis` command, to tell what a symbol represents. You may choose to specify a location or type a symbol (or even a whole word) to explain. Specifying a location

is done by moving the cursor to a particular spot on the map and then pressing one of '`.`', '`,`', '`;`', or '`:`'. '`.`' will explain the symbol at the chosen location, conditionally check for "`More info?`" depending upon whether the '*help*' option is on, and then you will be asked to pick another location; '`,`' will explain the symbol but skip any additional information, then let you pick another location; '`;`' will skip additional info and also not bother asking you to choose another location to examine; '`:`' will show additional info, if any, without asking for confirmation. When picking a location, pressing the `ESC` key will terminate this command, or pressing '`?`' will give a brief reminder about how it works.
If the *autodescribe* option is on, a short description of what you see at each location is shown as you move the cursor. Typing '`#`' while picking a location will toggle that option on or off. The *whatis_coord* option controls whether the short description includes map coordinates.
Specifying a name rather than a location always gives any additional information available about that name.
You may also request a description of nearby monsters, all monsters currently displayed, nearby objects, or all objects. The *whatis_coord* option controls which format of map coordinate is included with their descriptions.

`&` Tell what a command does.

`<` Go up to the previous level (if you are on a staircase or ladder).

`>` Go down to the next level (if you are on a staircase or ladder).

`[yuhjklbn]` Go one step in the direction indicated (see Figure 3). If you sense or remember a monster there, you will fight the monster instead. Only these one-step movement commands cause you to fight monsters; the others (below) are "safe."

```
y  k  u          7  8  9
 \ | /            \ | /
h- . -l          4- . -6
 / | \            / | \
b  j  n          1  2  3
```

(if *number_pad* set)

Figure 3

`[YUHJKLBN]` Go in that direction until you hit a wall or run into something.

`m[yuhjklbn]` Prefix: move without picking up objects or fighting (even if you remember a monster there).
A few non-movement commands use the '`m`' prefix to request operating via menu (to temporarily override the *menustyle:Traditional* option). Primarily useful for '`,`' (pickup) when there is only one class of objects present (where there won't be any "what kinds of objects?" prompt, so no opportunity to answer '`m`' at that prompt).
The prefix will make "`#travel`" command show a menu of interesting targets in sight. It can also be used with the '\' (known, show a list of all discovered objects) and the '`` ` ``' (knownclass, show a list of discovered objects in a particular class) commands to offer a menu of several sorting alternatives (which sets a new value for the *sortdiscoveries* option); also for "`#vanquished`" and "`#genocided`" commands to offer a sorting menu.
A few other commands (eat food, offer sacrifice, apply tinning-kit, drink/quaff, dip, tip container) use the '`m`' prefix to skip checking for applicable objects on the floor and go straight to checking inventory, or (for "`#loot`" to remove a saddle), skip containers and go straight to adjacent monsters.
In debug mode (aka "wizard mode"), the '`m`' prefix may also be used with the "`#teleport`" and "`#wizlevelport`" commands.

`F[yuhjklbn]` Prefix: fight a monster (even if you only guess one is there).

`g[yuhjklbn]` Prefix: Move until something interesting is found.

`G[yuhjklbn]` or `<Control>+[yuhjklbn]` Prefix: Similar to '`g`', but forking of corridors is not considered interesting.
Note: `<Control>+<key>` means holding the `<Control>` or `<Ctrl>` key down like `<Shift>` while typing and releasing `<key>`, then releasing `<Control>`. `^<key>` is used as shorthand elsewhere in the Guidebook to mean the same thing. Control characters are case-insensitive so `^x` and `^X` are the same.

`M[yuhjklbn]` Old versions supported '`M`' as a movement prefix which combined the effect of '`m`' with `<Control>+<direction>`. That is no longer supported as a prefix but similar effect can be achieved by using `m` and `G<direction>` in combination. `m` can also be used in combination with `g<direction>`, `<Control>+<direction>`, or `<Shift>+<direction>`.

`_` Travel to a map location via a shortest-path algorithm.
The shortest path is computed over map locations the hero knows about (e.g. seen or previously traversed). If there is no known path, a guess is made instead. Stops on most of the same conditions as the '`G`' command, but without picking up objects, so implicitly forces the '`m`' prefix. For ports with mouse support, the command is also invoked when a mouse-click takes place on a location other than the current position.

`.` Wait or rest, do nothing for one turn. Precede with the '`m`' prefix to wait for a turn even next to a hostile monster, if *safe_wait* is on.

`a` Apply (use) a tool (pick-axe, key, lamp ...).
If used on a wand, that wand will be broken, releasing its magic in the process. Confirmation is required.

`A` Remove one or more worn items, such as armor.
Use '`T`' (take off) to take off only one piece of armor or '`R`' (remove) to take off only one accessory.

`^A` Repeat the previous command.

`c` Close a door.

`C` Call (name) a monster, an individual object, or a type of object.
Same as extended command "`#name`".

`^C` Panic button. Quit the game.

`d` Drop something.
For example `d7a` — drop seven items of object *a*.

`D` Drop several things.
In answer to the question
"`What kinds of things do you want to drop? [!%= BUCXPaium]`"
you should type zero or more object symbols possibly followed by '`a`' and/or '`i`' and/or '`u`' and/or '`m`'. In addition, one or more of the blessed/uncursed/cursed groups may be typed.
`DB` — drop all objects known to be blessed.
`DU` — drop all objects known to be uncursed.
`DC` — drop all objects known to be cursed.
`DX` — drop all objects of unknown B/U/C status.
`DP` — drop objects picked up last.
`Da` — drop all objects, without asking for confirmation.
`Di` — examine your inventory before dropping anything.
`Du` — drop only unpaid objects (when in a shop).
`Dm` — use a menu to pick which object(s) to drop.
`D%u` — drop only unpaid food. The last example shows a combination. There are four

categories of object filtering: class ('!' for potions, '?' for scrolls, and so on), shop status ('u' for unpaid, in other words, owned by the shop), bless/curse state ('B', 'U', 'C', and 'X' as shown above), and novelty ('P', recently picked up items; controlled by picking up or dropping things rather than by any time factor).
If you specify more than one value in a category (such as "!?" for potions and scrolls or "BU" for blessed and uncursed), an inventory object will meet the criteria if it matches any of the specified values (so "!?" means '!' or '?'). If you specify more than one category, an inventory object must meet each of the category criteria (so "%u" means class '%' and unpaid 'u'). Lastly, you may specify multiple values within multiple categories: "!?BU" will select all potions and scrolls which are known to be blessed or uncursed. (In versions prior to 3.6, filter combinations behaved differently.)

^D — Kick something (usually a door).

e — Eat food.
Normally checks for edible item(s) on the floor, then if none are found or none are chosen, checks for edible item(s) in inventory. Precede 'e' with the 'm' prefix to bypass attempting to eat anything off the floor.
If you attempt to eat while already satiated, you might choke to death. If you risk it, you will be asked whether to "continue eating?" *if you survive the first bite.* You can set the *paranoid_confirmation:eating* option to require a response of "yes" instead of just 'y'.

E — Engrave a message on the floor.
E- — write in the dust with your fingers.
Engraving the word "Elbereth" will cause most monsters to not attack you hand-to-hand (but if you attack, you will rub it out); this is often useful to give yourself a breather.

f — Fire (shoot or throw) one of the objects placed in your quiver (or quiver sack, or that you have at the ready). You may select ammunition with a previous 'Q' command, or let the computer pick something appropriate if *autoquiver* is true. If your wielded weapon has the throw-and-return property, your quiver is empty, and *autoquiver* is false, you will throw that wielded weapon instead of filling the quiver. This will also automatically use a polearm if wielded. If *fireassist* is true, firing will automatically try to wield a launcher (for example, a bow or a sling) matching the ammo in the quiver; this might take multiple turns, and get interrupted by a monster. Remember to swap back to your main melee weapon afterwards.
See also 't' (throw) for more general throwing and shooting.

i — List your inventory (everything you're carrying).

I — List selected parts of your inventory, usually be specifying the character for a particular set of objects, like '[' for armor or '!' for potions.
I* — list all gems in inventory;
Iu — list all unpaid items;
Ix — list all used up items that are on your shopping bill;
IB — list all items known to be blessed;
IU — list all items known to be uncursed;
IC — list all items known to be cursed;
IX — list all items whose bless/curse status is unknown;
IP — list items picked up last;
I$ — count your money.

o — Open a door.

O — Set options.
A menu showing the current option values will be displayed. You can change most

values simply by selecting the menu entry for the given option (ie, by typing its letter or clicking upon it, depending on your user interface). For the non-boolean choices, a further menu or prompt will appear once you've closed this menu. The available options are listed later in this Guidebook. Options are usually set before the game rather than with the '`O`' command; see the section on options below. Precede `O` with the `m` prefix to show advanced options.

`^O` Show overview.
Shortcut for "`#overview`": list interesting dungeon levels visited.
(Prior to 3.6.0, '`^O`' was a debug mode command which listed the placement of all special levels. Use "`#wizwhere`" to run that command.)

`p` Pay your shopping bill.

`P` Put on an accessory (ring, amulet, or blindfold).
This command may also be used to wear armor. The prompt for which inventory item to use will only list accessories, but choosing an unlisted item of armor will attempt to wear it. (See the '`W`' command below. It lists armor as the inventory choices but will accept an accessory and attempt to put that on.)

`^P` Repeat previous message.
Subsequent `^P`'s repeat earlier messages. For some interfaces, the behavior can be varied via the *msg_window* option.

`q` Quaff (drink) something (potion, water, etc).
When there is a fountain or sink present, it asks whether to drink from that. If that is declined, then it offers a chance to choose a potion from inventory. Precede `q` with the `m` prefix to skip asking about drinking from a fountain or sink.

`Q` Select an object for your quiver, quiver sack, or just generally at the ready (only one of these is available at a time). You can then throw this (or one of these) using the '`f`' command.

`r` Read a scroll or spellbook.

`R` Remove a worn accessory (ring, amulet, or blindfold).
If you're wearing more than one, you'll be prompted for which one to remove. When you're only wearing one, then by default it will be removed without asking, but you can set the *paranoid_confirmation:Remove* option to require a prompt.
This command may also be used to take off armor. The prompt for which inventory item to remove only lists worn accessories, but an item of worn armor can be chosen. (See the '`T`' command below. It lists armor as the inventory choices but will accept an accessory and attempt to remove it.)

`^R` Redraw the screen.

`s` Search for secret doors and traps around you. It usually takes several tries to find something. Precede with the '`m`' prefix to wait for a turn even next to a hostile monster, if *safe_wait* is on.
Can also be used to figure out whether there is still a monster at an adjacent "remembered, unseen monster" marker.

`S` Save the game (which suspends play and exits the program). The saved game will be restored automatically the next time you play using the same character name.
In normal play, once a saved game is restored the file used to hold the saved data is deleted. In explore mode, once restoration is accomplished you are asked whether to keep or delete the file. Keeping the file makes it feasible to play for a while then quit without saving and later restore again.
There is no "save current game state and keep playing" command, not even in explore mode where saved game files can be kept and re-used.

`t` Throw an object or shoot a projectile.
There's no separate "shoot" command. If you "throw" an arrow while wielding a bow, you are shooting that arrow and any weapon skill bonus or penalty for bow applies. If you "throw" an arrow while not wielding a bow, you are throwing it by hand and it will generally be less effective than when shot.
See also '`f`' (fire) for throwing or shooting an item pre-selected via the '`Q`' (quiver) command, with some extra assistance.

`T` Take off armor.
If you're wearing more than one piece, you'll be prompted for which one to take off. (Note that this treats a cloak covering a suit and/or a shirt, or a suit covering a shirt, as if the underlying items weren't there.) When you're only wearing one, then by default it will be taken off without asking, but you can set the *paranoid_confirmation:Remove* option to require a prompt.
This command may also be used to remove accessories. The prompt for which inventory item to take off only lists worn armor, but a worn accessory can be chosen. (See the '`R`' command above. It lists accessories as the inventory choices but will accept an item of armor and attempt to take it off.)

`^T` Teleport, if you have the ability.

`v` Display a list of significant events in the current game, also shown by `#chronicle`.
'`v`' used to display the game's version number. Use '`V`' for that now.

`V` Display version number.
'`V`' used to display a summary of the game's history. Still available via `#history`.

`w` Wield weapon.
`w-` — wield nothing, use your bare (or gloved) hands.
Some characters can wield two weapons at once; use the '`X`' command (or the "`#twoweapon`" extended command) to do so.

`W` Wear armor.
This command may also be used to put on an accessory (ring, amulet, or blindfold). The prompt for which inventory item to use will only list armor, but choosing an unlisted accessory will attempt to put it on. (See the '`P`' command above. It lists accessories as the inventory choices but will accept an item of armor and attempt to wear it.)

`x` Exchange your wielded weapon with the item in your alternate weapon slot.
The latter is used as your secondary weapon when engaging in two-weapon combat. Note that if one of these slots is empty, the exchange still takes place.

`X` Toggle two-weapon combat, if your character can do it. Also available via the "`#twoweapon`" extended command.
(In versions prior to 3.6 this keystroke ran the command to switch from normal play to "explore mode", also known as "discovery mode", which has now been moved to "`#exploremode`" and `M-X`.)

`^X` Display basic information about your character.
Displays name, role, race, gender (unless role name makes that redundant, such as `Caveman` or `Priestess`), and alignment, along with your patron deity and his or her opposition. It also shows most of the various items of information from the status line(s) in a less terse form, including several additional things which don't appear in the normal status display due to space considerations.
In normal play, that's all that '`^X`' displays. In explore mode, the role and status feedback is augmented by the information provided by *enlightenment* magic.

`z` Zap a wand.

	`z.` — to aim at yourself, use '`.`' for the direction.
`Z`	Zap (cast) a spell. `Z.` — to cast at yourself, use '`.`' for the direction.
`^Z`	Suspend the game (UNIX versions with job control only). See "#suspend" below for more details.
`:`	Look at what is here.
`;`	Show what type of thing a visible symbol corresponds to.
`,`	Pick up some things from the floor beneath you. May be preceded by '`m`' to force a selection menu.
`@`	Toggle the *autopickup* option on and off.
`^`	Ask for the type of an adjacent trap you found earlier.
`)`	Tell what weapon you are wielding.
`]`	Tell what armor you are wearing.
`=`	Tell what rings you are wearing.
`"`	Tell what amulet you are wearing.
`(`	Tell what tools you are using.
`*`	Tell what equipment you are using. Combines the preceding five type-specific commands into one.
`$`	Report the gold you're carrying, possibly shop credit and/or debt too.
`+`	List the spells you know. Using this command, you can also rearrange the order in which your spells are listed, either by sorting the entire list or by picking one spell from the menu then picking another to swap places with it. Swapping pairs of spells changes their casting letters, so the change lasts after the current '`+`' command finishes. Sorting the whole list is temporary. To make the most recent sort order persist beyond the current '`+`' command, choose the sort option again and then pick "reassign casting letters". (Any spells learned after that will be added to the end of the list rather than be inserted into the sorted ordering.)
`\`	Show what types of objects have been discovered. May be preceded by '`m`' to select preferred display order.
`` ` ``	Show discovered types for one class of objects. May be preceded by '`m`' to select preferred display order.
`|`	If persistent inventory display is supported and enabled (with the *perm_invent* option), interact with it instead of with the map. Allows scrolling with the menu_first_page, menu_previous_page, menu_next_page, and menu_last_page keys ('`^`', '`<`', '`>`', '`|`' by default). Some interfaces also support menu_shift_left and menu_shift_right keys ('{' and '}' by default). Use the *Return* (aka *Enter*) or *Escape* key to resume play.
`!`	Escape to a shell. See "#shell" below for more details.
`Del`	Show map without obstructions. You can view the explored portion of the current level's map without monsters; without monsters and objects; or without monsters, objects, and traps. The `<del>` key is also shown as `<delete>` on some keyboards or `<rubout>` on others. It is sometimes displayed as `^?` even though that is not an actual control character. Many terminals have an option to swap the `<delete>` and `<backspace>` keys, so typing the `<del>` key might not execute this command. If that happens, you can use the extended command "`#terrain`" instead.

`#` Perform an extended command.
As you can see, the authors of *NetHack* used up all the letters, so this is a way to introduce the less frequently used commands. What extended commands are available depends on what features the game was compiled with.

`#adjust` Adjust inventory letters (most useful when the *fixinv* option is "on"). Autocompletes. Default key is '`M-a`'.
This command allows you to move an item from one particular inventory slot to another so that it has a letter which is more meaningful for you or that it will appear in a particular location when inventory listings are displayed. You can move to a currently empty slot, or if the destination is occupied—and won't merge—the item there will swap slots with the one being moved. "`#adjust`" can also be used to split a stack of objects; when choosing the item to adjust, enter a count prior to its letter. Adjusting without a count used to collect all compatible stacks when moving to the destination. That behavior has been changed; to gather compatible stacks, "`#adjust`" a stack into its own inventory slot. If it has a name assigned, other stacks with the same name or with no name will merge provided that all their other attributes match. If it does not have a name, only other stacks with no name are eligible. In either case, otherwise compatible stacks with a different name will not be merged. This contrasts with using "`#adjust`" to move from one slot to a different slot. In that situation, moving (no count given) a compatible stack will merge if either stack has a name when the other doesn't and give that name to the result, while splitting (count given) will ignore the source stack's name when deciding whether to merge with the destination stack.

`#annotate` Allows you to specify one line of text to associate with the current dungeon level. All levels with annotations are displayed by the "`#overview`" command. Autocompletes. Default key is '`M-A`', and also '`^N`' if *number_pad* is on.
Preceding #annotate with the '`m`' prefix is the same as #overview with the prefix.

`#apply` Apply (use) a tool such as a pick-axe, a key, or a lamp. Default key is '`a`'.
If the tool used acts on items on the floor, using the '`m`' prefix skips those items.
If used on a wand, that wand will be broken, releasing its magic in the process. Confirmation is required.

`#attributes` Show your attributes. Default key is '`^X`'.

`#autopickup` Toggle the *autopickup* option. Default key is '`@`'.

`#bugreport` Bring up a browser window to submit a report to the *NetHack Development Team*. Can be disabled at the time the program is built; when enabled, CRASHREPORTURL must be set in the system configuration file.

`#call` Call (name) a monster, or an object in inventory, on the floor, or in the discoveries list, or add an annotation for the current level (same as "`#annotate`"). Default key is '`C`'.

`#cast` Cast a spell. Default key is '`Z`'.

`#chat` Talk to someone. Default key is '`M-c`'.

`#chronicle` Show a list of important game events. Default key is '`v`'.

`#close` Close a door. Default key is '`c`'.

`#conduct` List voluntary challenges you have maintained. Autocompletes. Default key is '`M-C`'. See the section below entitled "Conduct" for details.

`#debugfuzzer` Start the fuzz tester. Debug mode only.

`#dip` Dip an object into something. Autocompletes. Default key is '`M-d`'.
The `m` prefix skips dipping into a fountain or pool if there is one at your location.

`#down`	Go down a staircase. Default key is '`>`'.
`#drop`	Drop an item. Default key is '`d`'.
`#droptype`	Drop specific item types. Default key is '`D`'.
`#eat`	Eat something. Default key is '`e`'. The '`m`' prefix skips eating items on the floor.
`#engrave`	Engrave writing on the floor. Default key is '`E`'.
`#enhance`	Advance or check weapon and spell skills. Autocompletes. Default key is '`M-e`'.
`#exploremode`	Switch from normal play to non-scoring explore mode. Default key is '`M-X`'. Requires confirmation; default response is '`n`' (no). To really switch to explore mode, respond with '`y`'. You can set the *paranoid_confirmation:quit* option to require a response of "`yes`" instead.
`#fight`	Prefix key to force fight a direction, even if you see nothing to fight there. Default key is '`F`', or '`-`' with *number_pad*
`#fire`	Fire ammunition from quiver, possibly autowielding a launcher, or hit with a wielded polearm. Default key is '`f`'.
`#force`	Force a lock. Autocompletes. Default key is '`M-f`'.
`#genocided`	List any monster types which have been genocided. In explore mode and debug mode it also shows types which have become extinct. The display order is the same as is used by `#vanquished`. The '`m`' prefix brings up a menu of available sorting orders, and doing that for either `#genocided` or `#vanquished` changes the order for both. If the sorting order is "count high to low" or "count low to high" (which are applicable for `#vanquished`), that will be ignored for `#genocided` and alphabetical will be used instead. The menu omits those two choices when used for `#genocide`. Autocompletes. Default key is '`M-g`'.
`#glance`	Show what type of thing a map symbol corresponds to. Default key is '`;`'.
`#help`	Show the help menu. Default key is '`?`', and also '`h`' if *number_pad* is on.
`#herecmdmenu`	Show a menu of possible actions directed at your current location. The menu is limited to a subset of the likeliest actions, not an exhaustive set of all possibilities. Autocompletes. If mouse support is enabled and the *herecmd_menu* option is On, clicking on the hero (or steed when mounted) will execute this command.
`#history`	Show a summary of the game's development.
`#inventory`	Show your inventory. Default key is '`i`'.
`#inventtype`	Inventory specific item types. Default key is '`I`'.
`#invoke`	Invoke an object's special powers. Autocompletes. Default key is '`M-i`'.
`#jump`	Jump to another location. Autocompletes. Default key is '`M-j`', and also '`j`' if *number_pad* is on.
`#kick`	Kick something. Default key is '`^D`', and also '`k`' if *number_pad* is on.
`#known`	Show what object types have been discovered. Default key is '\'. The '`m`' prefix allows assigning a new value to the *sortdiscoveries* option to control the order in which the discoveries are displayed.
`#knownclass`	Show discovered types for one class of objects. Default key is '`‘`'. The '`m`' prefix operates the same as for `#known`.
`#levelchange`	Change your experience level. Autocompletes. Debug mode only.
`#lightsources`	Show mobile light sources. Autocompletes. Debug mode only.
`#look`	Look at what is here, under you. Default key is '`:`'.

`#lookaround` Describe what you can see, or remember, of your surroundings.

`#loot` Loot a box or bag on the floor beneath you, or the saddle from a steed standing next to you. Autocompletes. Precede with the '`m`' prefix to skip containers at your location and go directly to removing a saddle. Default key is '`M-l`', and also '`l`' if *number_pad* is on.

`#monster` Use a monster's special ability (when polymorphed into monster form). Autocompletes. Default key is '`M-m`'.

`#name` Name a monster, an individual object, or a type of object. Same as "`#call`". Autocompletes. Default keys are '`N`', '`M-n`', and '`M-N`'.

`#offer` Offer a sacrifice to the gods. Autocompletes. Default key is '`M-o`'.
You'll need to find an altar to have any chance at success. Corpses of recently killed monsters are the fodder of choice. The '`m`' prefix skips offering any items which are on the altar.

`#open` Open a door. Default key is '`o`'.

`#options` Show and change option settings. Default key is '`O`'. Precede with the `m` prefix to show advanced options.

`#optionsfull` Show advanced game option settings. No default key. Precede with the '`m`' prefix to execute the simpler options command. (Mainly useful if you use `BINDING=O:optionsfull` to switch '`O`' from simple options back to traditional advanced options.)

`#overview` Display information you've discovered about the dungeon. Any visited level with an annotation is included, and many things (altars, thrones, fountains, and so on; extra stairs leading to another dungeon branch) trigger an automatic annotation. If dungeon overview is chosen during end-of-game disclosure, every visited level will be included regardless of annotations.
Precede #overview with the '`m`' prefix to display the dungeon overview as a menu where you can select any visited level to add or remove an annotation without needing to return to that level. This will also force all visited levels to be displayed rather than just the "interesting" subset.
Autocompletes. Default keys are '`^O`', and '`M-O`'.

`#panic` Test the panic routine. Terminates the current game. Autocompletes. Debug mode only.
Asks for confirmation; default is '`n`' (no); continue playing. To really panic, respond with '`y`'. You can set the *paranoid_confirmation:quit* option to require a response of "`yes`" instead.

`#pay` Pay your shopping bill. Default key is '`p`'.

`#perminv` If persistent inventory display is supported and enabled (with the *perm_invent* option), interact with it instead of with the map. You'll be prompted for menu scrolling keystrokes such as '`>`' and '`<`'. Press `Return` or `Escape` to resume normal play. Default key is `|`.

`#pickup` Pick up things at the current location. Default key is '`,`'. The '`m`' prefix forces use of a menu.

`#polyself` Polymorph self. Autocompletes. Debug mode only.

`#pray` Pray to the gods for help. Autocompletes. Default key is '`M-p`'.
Praying too soon after receiving prior help is a bad idea. (Hint: entering the dungeon alive is treated as having received help. You probably shouldn't start off a new game by praying right away.) Since using this command by accident can cause trouble, there is an option to make you confirm your intent before praying. It is enabled by default, and you can reset the *paranoid_confirmation* option to disable it.

`#prevmsg` Show previously displayed game messages. Default key is '`^P`'.

`#puton` Put on an accessory (ring, amulet, etc). Default key is '`P`'.

`#quaff` Quaff (drink) something. Default key is '`q`'.
The `m` prefix skips drinking from a fountain or sink if there is one at your location.

`#quit` Quit the program without saving your game. Autocompletes.
Since using this command by accident would throw away the current game, you are asked to confirm your intent before quitting. Default response is '`n`' (no); continue playing. To really quit, respond with '`y`'. You can set the *paranoid_confirmation:quit* option to require a response of "`yes`" instead.

`#quiver` Select ammunition for quiver. Default key is '`Q`'.

`#read` Read a scroll, a spellbook, or something else. Default key is '`r`'.

`#redraw` Redraw the screen. Default key is '`^R`', and also '`^L`' if *number_pad* is on.

`#remove` Remove an accessory (ring, amulet, etc). Default key is '`R`'.

`#repeat` Repeat the previous command. Default key is '`^A`'.

`#reqmenu` Prefix key to modify the behavior or request menu from some commands. Prevents autopickup when used with movement commands. Default key is '`m`'.

`#retravel` Travel to a previously selected travel destination. Default key is '`C-_`'. See also `#travel`.

`#ride` Ride (or stop riding) a saddled creature. Autocompletes. Default key is '`M-R`'.

`#rub` Rub a lamp or a stone. Autocompletes. Default key is '`M-r`'.

`#run` Prefix key to run towards a direction. Default key is '`G`' when *number_pad* is off, '`5`' when *number_pad* is set to 1 or 3, otherwise '`M-5`' when it is set to 2 or 4.

`#rush` Prefix key to rush towards a direction. Default key is '`g`' when *number_pad* is off, '`M-5`' when *number_pad* is set to 1 or 3, otherwise '`5`' when it is set to 2 or 4.

`#save` Save the game and exit the program. Default key is '`S`'.

`#saveoptions` Save configuration options to the config file. This will overwrite the file, removing all comments, so if you have manually edited the config file, don't use this.

`#search` Search for traps and secret doors around you. Default key is '`s`'.

`#seeall` Show all equipment in use. Default key is '`*`'.
Will display in-use items in a menu even when there is only one.

`#seeamulet` Show the amulet currently worn. Default key is '`"`'.
Using the '`m`' prefix will force the display of a worn amulet in a menu rather than with just a message.

`#seearmor` Show the armor currently worn. Default key is '`[`'.
Will display worn armor in a menu even when there is only thing worn.

`#seerings` Show the ring(s) currently worn. Default key is '`=`'. Will display worn rings in a menu if there are two (or there is just one and is a meat ring rather than a "real" ring). Use the '`m`' prefix to force a menu for one ring.

`#seetools` Show the tools currently in use. Default key is '`(`'. Will display the result in a message if there is one tool in use (worn blindfold or towel or lenses, lit lamp(s) and/or candle(s), leashes attached to pets). Will display a menu if there are more than one or if the command is preceded by the '`m`' prefix.

`#seeweapon` Show the weapon currently wielded. Default key is '`)`'. If dual-wielding, a separate message about the secondary weapon will be given. Using the '`m`' prefix will force a menu and it will include primary weapon, alternate weapon even when not dual-wielding, and also whatever is currently assigned to the quiver slot.

`#shell` Do a shell escape, switching from NetHack to a subprocess. Can be disabled at the time the program is built. When enabled, access for specific users can be controlled by the system configuration file. Use the shell command '`exit`' to return to the game. Default key is '`!`'.

`#showgold` Report the gold in your inventory, including gold you know about in containers you're carrying. If you are inside a shop, report any credit or debt you have in that shop. Default key is '`$`'.

`#showspells` List and reorder known spells. Default key is '`+`'.

`#showtrap` Describe an adjacent trap, possibly covered by objects or a monster. To be eligible, the trap must already be discovered. (The "`#terrain`" command can display your map with all objects and monsters temporarily removed, making it possible to see all discovered traps.) Default key is '`^`'.

`#sit` Sit down. Autocompletes. Default key is '`M-s`'.

`#stats` Show memory usage statistics. Autocompletes. Debug mode only.

`#suspend` Suspend the game, switching from NetHack to the terminal it was started from without performing save-and-exit. Can be disabled at the time the program is built. When enabled, mainly useful for *tty* and *curses* interfaces on UNIX. Use the shell command '`fg`' to return to the game. Default key is '`^Z`'.

`#swap` Swap wielded and secondary weapons. Default key is '`x`'.

`#takeoff` Take off one piece of armor. Default key is '`T`'.

`#takeoffall` Remove all armor. Default key is '`A`'.

`#teleport` Teleport around the level. Default key is '`^T`'.

`#terrain` Show map without obstructions. In normal play you can view the explored portion of the current level's map without monsters; without monsters and objects; or without monsters, objects, and traps.
If there are visible clouds of gas in view, they are treated like traps when deciding whether to show them or the floor underneath them.
In explore mode, you can choose to view the full map rather than just its explored portion. In debug mode there are additional choices.
Autocompletes. Default key is '`<del>`' or '`<delete>`' (see *Del* above).

`#therecmdmenu` Show a menu of possible actions directed at a location next to you. The menu is limited to a subset of the likeliest actions, not an exhaustive set of all possibilities. Autocompletes.

`#throw` Throw something. Default key is '`t`'.

`#timeout` Look at the timeout queue. Autocompletes. Debug mode only.

`#tip` Tip over a container (bag or box) to pour out its contents. When there are containers on the floor, the game will prompt to pick one of them or "tip something being carried".
If the latter is chosen, there will be another prompt for which item from inventory to tip. The '`m`' prefix makes the command skip containers on the floor and pick one from inventory, except for the special case of *menustyle:Traditional* with two or more containers present; that situation will start with the floor container menu.
Autocompletes. Default key is '`M-T`'.

`#toggle` Toggle a boolean option on or off. Requires a parameter in parenthesis, the name of the option to toggle. The option must be settable in-game.
For example:

```
BIND=':toggle(price_quotes)
BIND=@:toggle(autopickup)
```

`#travel` Travel to a specific location on the map. Default key is '`_`'. Using the "request menu" prefix shows a menu of interesting targets in sight without asking to move the cursor. When picking a target with cursor and the *autodescribe* option is on, the top line will show "(no travel path)" if your character does not know of a path to that location. See also `#retravel`.

`#turn` Turn undead away. Autocompletes. Default key is '`M-t`'.

`#twoweapon` Toggle two-weapon combat on or off. Autocompletes. Default key is '`X`', and also '`M-2`' if *number_pad* is off.
Note that you must use suitable weapons for this type of combat, or it will be automatically turned off.

`#untrap` Untrap something (trap, door, or chest). Default key is '`M-u`', and '`u`' if *number_pad* is on.
In some circumstances it can also be used to rescue trapped monsters.

`#up` Go up a staircase. Default key is '`<`'.

`#vanquished` List vanquished monsters by type and count.
Note that the vanquished monsters list includes all monsters killed by traps and each other as well as by you, and omits any which got removed from the game without being killed (perhaps by genocide, or by a mollified shopkeeper dismissing summoned Kops) or were already corpses when placed on the map.
Using the "request menu" prefix prior to #vanquished brings up a menu of sorting orders available (provided that the vanquished monsters list contains at least two types of monsters). Whichever ordering is picked gets assigned to the *sortvanquished* option so is remembered for subsequent #vanquished requests. The `#genocided` command shares this sorting order.
During end-of-game disclosure, when asked whether to show vanquished monsters answering '`a`' will let you choose from the sort menu.
Autocompletes. Default key is '`M-V`'.

`#version` Print compile time options for this version of *NetHack*.
The second paragraph lists the user interface(s) that are included. If there are more than one, you can use the *windowtype* option in your run-time configuration file to select the one you want.
Autocompletes. Default key is '`M-v`'.

`#versionshort` Show the program's version number, plus the date and time that the running copy was built from sources (not the version's release date). Default key is '`V`'.

`#vision` Show vision array. Autocompletes. Debug mode only.

`#wait` Rest one move while doing nothing. Default key is '`.`', and also '` `' if *rest_on_space* is on.

`#wear` Wear a piece of armor. Default key is '`W`'.

`#whatdoes` Tell what a key does. Default key is '`&`'.

`#whatis` Show what type of thing a symbol corresponds to. Default key is '`/`'.

`#wield` Wield a weapon. Default key is '`w`'.

`#wipe` Wipe off your face. Autocompletes. Default key is '`M-w`'.

`#wizborn` Show monster birth, death, genocide, and extinct statistics. Debug mode only.

`#wizbury` Bury objects under and around you. Autocompletes. Debug mode only.

`#wizcast` Cast any spell. Debug mode only.

`#wizdetect` Reveal hidden things (secret doors or traps or unseen monsters) within a modest radius. No time elapses. Autocompletes. Debug mode only. Default key is '`^E`'.

`#wizgenesis`	Create a monster. May be prefixed by a count to create more than one. Autocompletes. Debug mode only. Default key is '`^G`'.
`#wizidentify`	Identify all items in inventory. Autocompletes. Debug mode only. Default key is '`^I`'.
`#wizintrinsic`	Set one or more intrinsic attributes. Autocompletes. Debug mode only.
`#wizkill`	Remove monsters from play by just pointing at them. By default the hero gets credit or blame for killing the targets. Precede this command with the '`m`' prefix to override that. Autocompletes. Debug mode only.
`#wizlevelport`	Teleport to another level. Autocompletes. Debug mode only. Default key is '`^V`'.
`#wizmap`	Map the level. Autocompletes. Debug mode only. Default key is '`^F`'.
`#wizrumorcheck`	Verify rumor boundaries by displaying first and last true rumors and first and last false rumors. Also displays first, second, and last random engravings, epitaphs, and hallucinatory monsters. Autocompletes. Debug mode only.
`#wizseenv`	Show map locations' seen vectors. Autocompletes. Debug mode only.
`#wizsmell`	Smell monster. Autocompletes. Debug mode only.
`#wizwhere`	Show locations of special levels. Autocompletes. Debug mode only.
`#wizwish`	Wish for something. Autocompletes. Debug mode only. Default key is '`^W`'. Precede this command with the '`m`' prefix to show a wish history menu.
`#wmode`	Show wall modes. Autocompletes. Debug mode only.
`#zap`	Zap a wand. Default key is '`z`'.
`#?`	Help menu: get the list of available extended commands.

If your keyboard has a meta key (which, when pressed in combination with another key, modifies it by setting the 'meta' [8th, or 'high'] bit), you can invoke many extended commands by meta-ing the first letter of the command.

On *Windows* and *MS-DOS*, the 'Alt' key can be used in this fashion. On other systems, if typing 'Alt' plus another key transmits a two character sequence consisting of an `Escape` followed by the other key, you may set the *altmeta* option to have *NetHack* combine them into `meta+<key>`. (This combining action only takes place when NetHack is expecting a command to execute, not when accepting input to name something or to make a wish.)

Unlike control characters, where `^x` and `^X` denote the same thing, meta characters are case-sensitive: `M-x` and `M-X` represent different things. Some commands which can be run via a meta character require that the letter be capitalized because the lower-case equivalent is used for another command, so the three key combination `meta+Shift+letter` is needed.

`M-?`	`#?` (not supported by all platforms)
`M-2`	`#twoweapon` (unless the *number_pad* option is enabled)
`M-a`	`#adjust`
`M-A`	`#annotate`
`M-c`	`#chat`
`M-C`	`#conduct`
`M-d`	`#dip`
`M-e`	`#enhance`
`M-f`	`#force`
`M-g`	`#genocided`
`M-i`	`#invoke`

M-j	#jump
M-l	#loot
M-m	#monster
M-n	#name
M-o	#offer
M-O	#overview
M-p	#pray
M-r	#rub
M-R	#ride
M-s	#sit
M-t	#turn
M-T	#tip
M-u	#untrap
M-v	#version
M-V	#vanquished
M-w	#wipe
M-X	#exploremode

If the *number_pad* option is on, some additional letter commands are available:

h	#help
j	#jump
k	#kick
l	#loot
N	#name
u	#untrap

5 Rooms and corridors

Rooms and corridors in the dungeon are either lit or dark. Any lit areas within your line of sight will be displayed; dark areas are only displayed if they are within one space of you. Walls and corridors remain on the map as you explore them.

Secret corridors are hidden and appear to be solid rock. You can find them with the '`s`' (search) command when adjacent to them. Multiple search attempts may be needed. When searching is successful, secret corridors become ordinary open corridor locations. Mapping magic reveals secret corridors, so converts them into ordinary corridors and shows them as such.

Doorways

Doorways connect rooms and corridors. Some doorways have no doors; you can walk right through. Others have doors in them, which may be open, closed, or locked. To open a closed door, use the '`o`' (open) command; to close it again, use the '`c`' (close) command. By default the *autoopen* option is enabled, so simply attempting to walk onto a closed door's location will attempt to open it without needing '`o`'. Opening via *autoopen* will not work if you are *confused* or *stunned* or suffer from the *fumbling* attribute.

Open doors cannot be entered diagonally; you must approach them straight on, horizontally or vertically. Doorways without doors are not restricted in this fashion except on one particular level (described by "`#overview`" as "a primitive area").

Unlocking magic exists but usually won't be available early on. You can get through a locked door without magic by first using an unlocking tool with the '`a`' (apply) command, and then opening it. By default the *autounlock* option is also enabled, so if you attempt to open (via '`o`' or *autoopen*) a locked door while carrying an unlocking tool, you'll be asked whether to use it on the door's lock. Alternatively, you can break a closed door (whether locked or not) down by kicking it via the "`^D`" (kick) command. Kicking down a door destroys it and makes a lot of noise which might wake sleeping monsters.

Some closed doors are booby-trapped and will explode if an attempt is made to open (when unlocked) or unlock (when locked) or kick down. Like kicking, an explosion destroys the door and makes a lot of noise. The "`#untrap`" command can be used to search a door for traps but might take multiple attempts to find one. When one is found, you'll be asked whether to try to disarm it. If you accede, success will eliminate the trap but failure will set off the trap's explosion. (If you decline, you effectively forget that a trap was found there.)

Closed doors can be useful for shutting out monsters. Most monsters cannot open closed doors, although a few don't need to (for example, ghosts can walk through doors and fog clouds can flow under them). Some monsters who can open doors can also use unlocking tools. And some (giants) can smash doors.

Secret doors are hidden and appear to be ordinary wall (from inside a room) or solid rock (from outside). You can find them with the '`s`' (search) command but it might take multiple tries (possibly many tries if your luck is poor). Once found they are in all ways equivalent to normal doors. Mapping magic does not reveal secret doors.

Traps ('^')

There are traps throughout the dungeon to snare the unwary intruder. For example, you may suddenly fall into a pit and be stuck for a few turns trying to climb out (see below). A trap usually won't appear on your map until you trigger it by moving onto it, you see someone else trigger it, or you discover it with the '`s`' (search) command (multiple attempts are often needed; if your luck is poor, many attempts might be needed). *Wands of secret door detection* and the spell of *detect unseen* also reveal traps within a modest radius but only if the trap is also within line-of-sight (whether you can see at the time or not). There is also other magic which can reveal traps.

Monsters can fall prey to traps, too, which can potentially be used as a defensive strategy. Unfortunately traps can be harmful to your pet(s) as well. Monsters, including pets, usually will avoid moving onto a trap which is shown on your map if they have encountered that type of trap before.

Some traps such as pits, bear traps, and webs hold you in one place. You can escape by simply trying to move to an adjacent spot and repeat as needed; eventually you will get free.

Other traps can send you to different locations. Teleporters send you elsewhere on the same dungeon level. Level teleporters send you to a random dungeon level, the destination chosen from a few levels lower all the way to the top. These traps choose a new destination each time they're activated. Trap doors and holes also send you to another level, but one which is always below the current level. Usually that will be the next level down but it can be farther. Unlike (level) teleporters, the destination level of a particular trap door or hole is persistent, so falling into one will bring you to the same level each time—though not necessarily the same spot on the level. Magic portals behave similarly, but with some additional variation. Some portals are two-way and their remote destination is always the same: another portal which can take you back. Others are one-way and send you to a specific destination level but not necessarily to a specific location there.

There is a special multi-level branch of the dungeon with pre-mapped levels based on the classic computer game "*Sokoban*." In that game, you operate as a warehouse worker who pushes crates around obstacles to position them at designated locations. In NetHack, the goal is to push boulders into pits or holes until those traps have all been nullified, giving access to whatever is beyond them. In the Sokoban game, you can only move in the four cardinal compass directions, and a crate in its final destination blocks further access to that spot. In the Sokoban levels of NetHack, you can move diagonally (unless that would let you pass between two neighboring boulders) but you can only push boulders in the four

cardinal directions, and a boulder which fills a pit or hole removes both the boulder and the trap so opens up normal access to that spot. With careful foresight, it is possible to complete all of the levels according to the traditional rules of Sokoban. (Hint: to solve Sokoban puzzles, you often need to move things away from their eventual destinations in order to open up more room to maneuver.) Since NetHack does not support an *undo* capability, some allowances are permitted in case you get stuck. For example, each level has at least one extra boulder. Also, it is possible to drop everything in order to be able to squeeze into the same location as a boulder (and then presumably move past it), or to destroy a boulder with magic or tools, or to create new boulders with a *scroll of earth*. However, doing such things will lower your luck without any specific message given about that. See the *Conduct* section for information about getting feedback for your actions in Sokoban.

Stairs and ladders ('<', '>')

In general, each level in the dungeon will have a staircase going up ('<') to the previous level and another going down ('>') to the next level. There are some exceptions though. For instance, fairly early in the dungeon you will find a level with two down staircases, one continuing into the dungeon and the other branching into an area known as the Gnomish Mines. Those mines eventually hit a dead end, so after exploring them (if you choose to do so), you'll need to climb back up to the main dungeon.

When you traverse a set of stairs, or trigger a trap which sends you to another level, the level you're leaving will be deactivated and stored in a file on disk. If you're moving to a previously visited level, it will be loaded from its file on disk and reactivated. If you're moving to a level which has not yet been visited, it will be created (from scratch for most random levels, from a template for some "special" levels, or loaded from the remains of an earlier game for a "bones" level as briefly described below). Monsters are only active on the current level; those on other levels are essentially placed into stasis.

Ordinarily when you climb a set of stairs, you will arrive on the corresponding staircase at your destination. However, pets (see below) and some other monsters will follow along if they're close enough when you travel up or down stairs, and occasionally one of these creatures will displace you during the climb. When that occurs, the pet or other monster will arrive on the staircase and you will end up nearby.

Ladders serve the same purpose as staircases, and the two types of inter-level connections are nearly indistinguishable during game play.

Shops and shopping

Occasionally you will run across a room with a shopkeeper near the door and many items lying on the floor. You can buy items by picking them up and then using the 'p' command. You can inquire about the price of an item prior to picking it up by using the "`#chat`" command while standing on it. Using an item prior to paying for it will incur a charge, and the shopkeeper won't allow you to leave the shop until you have paid any debt you owe.

You can sell items to a shopkeeper by dropping them to the floor while inside a shop. You will either be offered an amount of gold and asked whether you're willing to sell, or you'll be told that the shopkeeper isn't interested (generally, your item needs to be compatible with the type of merchandise carried by the shop).

If you drop something in a shop by accident, the shopkeeper will usually claim ownership without offering any compensation. You'll have to buy it back if you want to reclaim it.

Shopkeepers sometime run out of money. When that happens, you'll be offered credit instead of gold when you try to sell something. Credit can be used to pay for purchases, but it is only good in the shop where it was obtained; other shopkeepers won't honor it. (If you happen to find a "credit card" in the dungeon, don't bother trying to use it in shops; shopkeepers will not accept it.)

The `$` command, which reports the amount of gold you are carrying, will also show current shop debt or credit, if any. The `Iu` command lists unpaid items (those which still belong to the shop) if you are carrying any. The `Ix` command shows an inventory-like display of any unpaid items which have been used up, along with other shop fees, if any.

Shop idiosyncrasies

Several aspects of shop behavior might be unexpected.

- The price of a given item can vary due to a variety of factors.
- A shopkeeper treats the spot immediately inside the door as if it were outside the shop.
- While the shopkeeper watches you like a hawk, he or she will generally ignore any other customers.
- If a shop is "closed for inventory," it will not open of its own accord.
- Shops do not get restocked with new items, regardless of inventory depletion.

Movement feedback

Moving around the map usually provides no feedback—other than drawing the hero at the new location—unless you step on an object or pile of objects, or on a trap, or attempt to move onto a spot where a monster is located. There are several options which can be used to augment the normal feedback.

The *pile_limit* option controls how many objects can be in a pile—sharing the same map location—for the game to state "there are objects here" instead of listing them. The default is `5`. Setting it to `1` would always give that message instead of listing any objects. Setting it to `0` is a special case which will always list all objects no matter how big a pile is. Note that the number refers to the count of separate stacks of objects present rather than the sum of the quantities of those stacks (so `7 arrows` or `25 gold pieces` will each count as 1 rather than as 7 and 25, respectively, and total to 2 when both are at the same location).

The `nopickup` command prefix (default '`m`') can be used before a movement direction to step on objects without attempting auto-pickup and without giving feedback about them.

The *mention_walls* option controls whether you get feedback if you try to walk into a wall or solid stone or off the edge of the map. Normally nothing happens (unless the hero is blind and no wall is shown, then the wall that is being bumped into will be drawn on the map). This option also gives feedback when rushing or running stops for some non-obvious reason.

The *mention_decor* option controls whether you get feedback when walking on "furniture." Normally stepping onto stairs or a fountain or an altar or various other things doesn't elicit anything unless it is covered by one or more objects so is obscured on the map. Setting this option to true will describe such things even when they aren't obscured. Doorless doorways and open doors aren't considered worthy of mention; closed doors (if you can move onto their spots) and broken doors are. Assuming that you're able to do so, moving onto water or lava or ice will give feedback if not yet on that type of terrain but not repeat it (unless there has been some intervening message) when moving from water to another water spot, or lava to lava, or ice to ice. Moving off of any of those back onto "normal" terrain will give one message too, unless there is feedback about one or more objects, in which case the back on land circumstance is implied.

The *confirm* and *safe_pet* options control what happens when you try to move onto a peaceful monster's spot or a tame one's spot.

The `nopickup` command prefix (default '`m`') is also the move-without-attacking prefix and can be used to try to step onto a visible monster's spot without the move being considered an attack (see the *Fighting* subsection of *Monsters* below). The '`fight`' command prefix (default '`F`'; also '`-`' if *number_pad* is on) can be used to force an attack, when guessing where an unseen monster is or when deliberately attacking a peaceful or tame creature.

The *run_mode* option controls how frequently the map gets redrawn when moving more than one step in a single command (so when rushing, running, or traveling).

Rogue level

One dungeon level (occurring in mid to late teens of the main dungeon) is a tribute to the ancestor game *hack*'s inspiration *rogue*.

It is usually displayed differently from other levels: possibly in characters instead of tiles, or without line-drawing symbols if already in characters; also, gold is shown as `*` rather than `$` and stairs are shown as `%` rather than `<` and `>`. There are some minor differences in actual game play: doorways lack doors; a scroll, wand, or spell of light used in a room lights up the whole room rather than within a radius around your character. And monsters represented by lower-case letters aren't randomly generated on the rogue level.

The slight strangeness of this level is a feature, not a bug....

6 Monsters

Monsters you cannot see are not displayed on the screen. Beware! You may suddenly come upon one in a dark place. Some magic items can help you locate them before they locate you (which some monsters can do very well).

The commands '`/`' and '`;`' may be used to obtain information about those monsters who are displayed on the screen. The command "`#name`" (by default bound to '`C`'), allows you to assign a name to a monster, which may be useful to help distinguish one from another when multiple monsters are present. Assigning a name which is just a space will remove any prior name.

The extended command "`#chat`" can be used to interact with an adjacent monster. There is no actual dialog (in other words, you don't get to choose what you'll say), but chatting with some monsters such as a shopkeeper or the Oracle of Delphi can produce useful results.

Fighting

If you see a monster and you wish to fight it, just attempt to walk into it. Many monsters you find will mind their own business unless you attack them. Some of them are very dangerous when angered. Remember: discretion is the better part of valor.

In most circumstances, if you attempt to attack a peaceful monster by moving into its location, you'll be asked to confirm your intent. By default an answer of '`y`' acknowledges that intent, which can be error prone if you're using '`y`' to move. You can set the *paranoid_confirmation:attack* option to require a response of "`yes`" instead.

If you can't see a monster (if it is invisible, or if you are blinded), the symbol '`I`' will be shown when you learn of its presence. If you attempt to walk into it, you will try to fight it just like a monster that you can see; of course, if the monster has moved, you will attack empty air. If you guess that the monster has moved and you don't wish to fight, you can use the '`m`' command to move without fighting; likewise, if you don't remember a monster but want to try fighting anyway, you can use the '`F`' command.

Your pet

You start the game with a little dog ('`d`'), kitten ('`f`'), or pony ('`u`'), which follows you about the dungeon and fights monsters with you. Like you, your pet needs food to survive. Dogs and cats usually feed themselves on fresh carrion and other meats; horses need vegetarian food which is harder to come by. If you're worried about your pet or want to train it, you can feed it, too, by throwing it food. A properly trained pet can be very useful under certain circumstances.

Your pet also gains experience from killing monsters, and can grow over time, gaining hit points and doing more damage. Initially, your pet may even be better at killing things than you, which makes pets useful for low-level characters.

Your pet will follow you up and down staircases if it is next to you when you move. Otherwise your pet will be stranded and may become wild. Similarly, when you trigger certain types of traps which alter

your location (for instance, a trap door which drops you to a lower dungeon level), any adjacent pet will accompany you and any non-adjacent pet will be left behind. Your pet may trigger such traps itself; you will not be carried along with it even if adjacent at the time.

Steeds

Some types of creatures in the dungeon can actually be ridden if you have the right equipment and skill. Convincing a wild beast to let you saddle it up is difficult to say the least. Many a dungeoneer has had to resort to magic and wizardry in order to forge the alliance. Once you do have the beast under your control however, you can easily climb in and out of the saddle with the "`#ride`" command. Lead the beast around the dungeon when riding, in the same manner as you would move yourself. It is the beast that you will see displayed on the map.

Riding skill is managed by the "`#enhance`" command. See the section on Weapon proficiency for more information about that.

Use the '`a`' (apply) command and pick a saddle in your inventory to attempt to put that saddle on an adjacent creature. If successful, it will be transferred to that creature's inventory.

Use the "`#loot`" command while adjacent to a saddled creature to try to remove the saddle from that creature. If successful, it will be transferred to your inventory.

Bones levels

You may encounter the shades and corpses of other adventurers (or even former incarnations of yourself!) and their personal effects. Ghosts are hard to kill, but easy to avoid, since they're slow and do little damage. You can plunder the deceased adventurer's possessions; however, they are likely to be cursed. Beware of whatever killed the former player; it is probably still lurking around, gloating over its last victory.

Persistence of Monsters

Monsters (a generic reference which also includes humans and pets) are only shown while they can be seen or otherwise sensed. Moving to a location where you can't see or sense a monster any more will result in it disappearing from your map, similarly if it is the one who moved rather than you.

However, if you encounter a monster which you can't see or sense—perhaps it is invisible and has just tapped you on the noggin—a special "remembered, unseen monster" marker will be displayed at the location where you think it is. That will persist until you have proven that there is no monster there, even if the unseen monster moves to another location or you move to a spot where the marker's location ordinarily wouldn't be seen any more.

7 Objects

When you find something in the dungeon, it is common to want to pick it up. In *NetHack*, this is accomplished by using the '`,`' command. If *autopickup* option is on, you will automatically pick up the object by walking over it, unless you move with the '`m`' prefix. If you're carrying too many items, *NetHack* will tell you so and you won't be able to pick up anything more. Otherwise, it will add the object(s) to your pack and tell you what you just picked up. As you add items to your inventory, you also add the weight of that object to your load. The amount that you can carry depends on your strength and your constitution. The stronger and sturdier you are, the less the additional load will affect you. There comes a point, though, when the weight of all of that stuff you are carrying around with you through the dungeon will encumber you. Your reactions will get slower and you'll burn calories faster, requiring food more frequently to cope with it. Eventually, you'll be so overloaded that you'll either have to discard some of what you're carrying or collapse under its weight. *NetHack* will tell you how badly you have loaded

yourself. If you are encumbered, one of the conditions `Burdened`, `Stressed`, `Strained`, `Overtaxed`, or `Overloaded` will be shown on the bottom line status display.

When you pick up an object, it is assigned an inventory letter. Many commands that operate on objects must ask you to find out which object you want to use. When *NetHack* asks you to choose a particular object you are carrying, you are usually presented with a list of inventory letters to choose from (see Commands, above).

Some objects, such as weapons, are easily differentiated. Others, like scrolls and potions, are given descriptions which vary according to type. During a game, any two objects with the same description are the same type. However, the descriptions will vary from game to game.

When you use one of these objects, if its effect is obvious, *NetHack* will remember what it is for you. If its effect isn't extremely obvious, you will be asked what you want to call this type of object so you will recognize it later. You can also use the "`#name`" command, for the same purpose at any time, to name all objects of a particular type or just an individual object. When you use "`#name`" on an object which has already been named, specifying a space as the value will remove the prior name instead of assigning a new one.

Curses and Blessings

Any object that you find may be cursed, even if the object is otherwise helpful. The most common effect of a curse is being stuck with (and to) the item. Cursed weapons weld themselves to your hand when wielded, so you cannot unwield them. Any cursed item you wear is not removable by ordinary means. In addition, cursed arms and armor usually, but not always, bear negative enchantments that make them less effective in combat. Other cursed objects may act poorly or detrimentally in other ways.

Objects can also be blessed instead. Blessed items usually work better or more beneficially than normal uncursed items. For example, a blessed weapon will do slightly more damage against demons.

Objects which are neither cursed nor blessed are referred to as uncursed. They could just as easily have been described as unblessed, but the uncursed designation is what you will see within the game. A "glass half full versus glass half empty" situation; make of that what you will.

There are magical means of bestowing or removing curses upon objects, so even if you are stuck with one, you can still have the curse lifted and the item removed. Priests and Priestesses have an innate sensitivity to this property in any object, so they can more easily avoid cursed objects than other character roles. Dropping objects onto an altar will reveal their bless or curse state provided that you can see them land.

An item with unknown status will be reported in your inventory with no prefix. An item which you know the state of will be distinguished in your inventory by the presence of the word `cursed`, `uncursed` or `blessed` in the description of the item. In some cases `uncursed` will be omitted as being redundant when enough other information is displayed. The *implicit_uncursed* option can be used to control this; toggle it off to have `uncursed` be displayed even when that can be deduced from other attributes.

Sometimes the bless or curse state of objects is referred to as their "`BUC`" attribute, for Blessed, Uncursed, or Cursed state, or "`BUCX`" for Blessed, Uncursed, Cursed, or unknown. (The term *beatitude* is occasionally used as well.)

Artifacts

Some objects have been imbued with special powers and are known as *Artifacts*. They have specific types (such as long sword or orcish dagger) and distinct names such as *Giantslayer* or *Grimtooth*. Artifact weapons typically do more damage than their ordinary counterparts. Some do extra damage against all monsters, others only against specific types of monsters so aren't better than regular weapons against other types. Some confer defensive capabilities when wielded or have other powers that aren't listed here.

You might find them simply lying on the floor, including but not limited to inside shops, or be granted as a reward for "`#offer`" on an altar to your patron deity. A few might be dropped by monsters, or

might be converted from an ordinary object of the same type via assigning the right name. Or you might wish for them, if you happen to be granted a wish, but such wishes can fail.

Some artifacts have a specific alignment, others don't. You won't obtain aligned ones that have a different alignment from yours via offering and might get a shock if you attempt to wish for any of those or find one and attempt to use it.

Each role has a distinct artifact that is contained in the *Quest* dungeon branch. These are commonly known as quest artifacts. All are aligned and most are non-weapons. They won't be found randomly.

The '\' and '`a' commands will list artifacts that you have fully identified (knowing the name and item type isn't sufficient).

Relics

There are three unique items that are named and have limited special powers but aren't classified as artifacts. Each is guarded by a particular monster and you'll need to collect all three for use late in the game. They are *the Bell of Opening*, *the Book of the Dead*, and *the Candelabrum of Invocation*. Their corresponding descriptions when not yet identified are silver bell, papyrus spellbook, and candelabrum.

Weapons (')')

Given a chance, most monsters in the Mazes of Menace will gratuitously try to kill you. You need weapons for self-defense (killing them first). Without a weapon, you do only 1–2 hit points of damage (plus bonuses, if any). Monk characters are an exception; they normally do more damage with bare (or gloved) hands than they do with weapons.

There are wielded weapons, like maces and swords, and thrown weapons, like arrows and spears. To hit monsters with a weapon, you must wield it and attack them, or throw it at them. You can simply elect to throw a spear. To shoot an arrow, you should first wield a bow, then throw the arrow. Crossbows shoot crossbow bolts. Slings hurl rocks and (other) stones (like gems).

Enchanted weapons have a "plus" (or "to hit enhancement" which can be either positive or negative) that adds to your chance to hit and the damage you do to a monster. The only way to determine a weapon's enchantment is to have it magically identified somehow. Most weapons are subject to some type of damage like rust. Such "erosion" damage can be repaired.

The chance that an attack will successfully hit a monster, and the amount of damage such a hit will do, depends upon many factors. Among them are: type of weapon, quality of weapon (enchantment and/or erosion), experience level, strength, dexterity, encumbrance, and proficiency (see below). The monster's armor class—a general defense rating, not necessarily due to wearing of armor—is a factor too; also, some monsters are particularly vulnerable to certain types of weapons.

Many weapons can be wielded in one hand; some require both hands. When wielding a two-handed weapon, you can not wear a shield, and vice versa. When wielding a one-handed weapon, you can have another weapon ready to use by setting things up with the 'x' command, which exchanges your primary (the one being wielded) and alternate weapons. And if you have proficiency in the "two weapon combat" skill, you may wield both weapons simultaneously as primary and secondary; use the 'X' command to engage or disengage that. Only some types of characters (barbarians, for instance) have the necessary skill available. Even with that skill, using two weapons at once incurs a penalty in the chance to hit your target compared to using just one weapon at a time.

There might be times when you'd rather not wield any weapon at all. To accomplish that, wield '-', or else use the 'A' command which allows you to unwield the current weapon in addition to taking off other worn items.

Those of you in the audience who are AD&D players, be aware that each weapon which existed in AD&D does roughly the same damage to monsters in *NetHack*. Some of the more obscure weapons (such as the *aklys*, *lucern hammer*, and *bec-de-corbin*) are defined in an appendix to *Unearthed Arcana*, an AD&D supplement.

Some interfaces support the `weaponstatus` option. When it is enabled, an extra status condition is displayed, describing the currently wielded weapon.

The commands to use weapons are '`w`' (wield), '`t`' (throw), '`f`' (fire), '`Q`' (quiver), '`x`' (exchange), '`X`' (twoweapon), and "`#enhance`" (see below).

Throwing and shooting

You can throw just about anything via the '`t`' command. It will prompt for the item to throw; picking '`?`' will list things in your inventory which are considered likely to be thrown, or picking '`*`' will list your entire inventory. After you've chosen what to throw, you will be prompted for a direction rather than for a specific target. The distance something can be thrown depends mainly on the type of object and your strength. Arrows can be thrown by hand, but can be thrown much farther and will be more likely to hit when thrown while you are wielding a bow.

Some weapons will return when thrown. A boomerang—provided it fails to hit anything—is an obvious example. If an aklys (thonged club) is thrown while it is wielded, it will return even when it hits something. A sufficiently strong hero can throw the warhammer *Mjollnir*; when thrown by a *Valkyrie* it will return too. However, aklyses and *Mjollnir* occasionally fail to return. Returning thrown objects occasionally fail to be caught, sometimes even hitting the thrower, but when caught they become re-wielded.

You can simplify the throwing operation by using the '`Q`' command to select your preferred "missile", then using the '`f`' command to throw it. You'll be prompted for a direction as above, but you don't have to specify which item to throw each time you use '`f`'. There is also an option, *autoquiver*, which has *NetHack* choose another item to automatically fill your quiver (or quiver sack, or have at the ready) when the inventory slot used for '`Q`' runs out. If your quiver is empty, *autoquiver* is false, and you are wielding a weapon which returns when thrown, you will throw that weapon instead of filling the quiver. The fire command also has extra assistance, if *fireassist* is on it will try to wield a launcher matching the ammo in the quiver.

Some characters have the ability to throw or shoot a volley of multiple items (from the same stack) in a single action. Knowing how to load several rounds of ammunition at once—or hold several missiles in your hand—and still hit a target is not an easy task. Rangers are among those who are adept at this task, as are those with a high level of proficiency in the relevant weapon skill (in bow skill if you're wielding one to shoot arrows, in crossbow skill if you're wielding one to shoot bolts, or in sling skill if you're wielding one to shoot stones). The number of items that the character has a chance to fire varies from turn to turn. You can explicitly limit the number of shots by using a numeric prefix before the '`t`' or '`f`' command. For example, "`2f`" (or "`n2f`" if using *number_pad* mode) would ensure that at most 2 arrows are shot even if you could have fired 3. If you specify a larger number than would have been shot ("`4f`" in this example), you'll just end up shooting the same number (3, here) as if no limit had been specified. Once the volley is in motion, all of the items will travel in the same direction; if the first ones kill a monster, the others can still continue beyond that spot.

Weapon proficiency

You will have varying degrees of skill in the weapons available. Weapon proficiency, or weapon skills, affect how well you can use particular types of weapons, and you'll be able to improve your skills as you progress through a game, depending on your role, your experience level, and use of the weapons.

For the purposes of proficiency, weapons have been divided up into various groups such as daggers, broadswords, and polearms. Each role has a limit on what level of proficiency a character can achieve for each group. For instance, wizards can become highly skilled in daggers or staves but not in swords or bows.

The "`#enhance`" extended command is used to review current weapons proficiency (also spell proficiency) and to choose which skill(s) to improve when you've used one or more skills enough to become eligible to do so. The skill rankings are "none" (sometimes also referred to as "restricted", because you

won't be able to advance), "unskilled", "basic", "skilled", and "expert". Restricted skills simply will not appear in the list shown by "`#enhance`". (Divine intervention might unrestrict a particular skill, in which case it will start at unskilled and be limited to basic.) Some characters can enhance their barehanded combat or martial arts skill beyond expert to "master" or "grand master".

Use of a weapon in which you're restricted or unskilled will incur a modest penalty in the chance to hit a monster and also in the amount of damage done when you do hit; at basic level, there is no penalty or bonus; at skilled level, you receive a modest bonus in the chance to hit and amount of damage done; at expert level, the bonus is higher. A successful hit has a chance to boost your training towards the next skill level (unless you've already reached the limit for this skill). Once such training reaches the threshold for that next level, you'll be told that you feel more confident in your skills. At that point you can use "`#enhance`" to increase one or more skills. Such skills are not increased automatically because there is a limit to your total overall skills, so you need to actively choose which skills to enhance and which to ignore.

Two-Weapon combat

Some characters can use two weapons at once. Setting things up to do so can seem cumbersome but becomes second nature with use. To wield two weapons, you need to use the "`#twoweapon`" command. But first you need to have a weapon in each hand. (Note that your two weapons are not fully equal; the one in the hand you normally wield with is considered primary and the other one is considered secondary. The most noticeable difference is after you stop—or before you begin, for that matter—wielding two weapons at once. The primary is your wielded weapon and the secondary is just an item in your inventory that's been designated as alternate weapon.)

If your primary weapon is wielded but your off hand is empty or has the wrong weapon, use the sequence '`x`', '`w`', '`x`' to first swap your primary into your off hand, wield whatever you want as secondary weapon, then swap them both back into the intended hands. If your secondary or alternate weapon is correct but your primary one is not, simply use '`w`' to wield the primary. Lastly, if neither hand holds the correct weapon, use '`w`', '`x`', '`w`' to first wield the intended secondary, swap it to off hand, and then wield the primary.

The whole process can be simplified via use of the *pushweapon* option. When it is enabled, then using '`w`' to wield something causes the currently wielded weapon to become your alternate weapon. So the sequence '`w`', '`w`' can be used to first wield the weapon you intend to be secondary, and then wield the one you want as primary which will push the first into secondary position.

When in two-weapon combat mode, using the '`X`' command toggles back to single-weapon mode. Throwing or dropping either of the weapons or having one of them be stolen or destroyed will also make you revert to single-weapon combat.

Armor ('`[`')

Lots of unfriendly things lurk about; you need armor to protect yourself from their blows. Some types of armor offer better protection than others. Your armor class is a measure of this protection. Armor class (AC) is measured as in AD&D, with 10 being the equivalent of no armor, and lower numbers meaning better armor. Each suit of armor which exists in AD&D gives the same protection in *NetHack*.

Here is a list of the armor class values provided by suits of armor:

dragon scale mail	1	plate mail	3
crystal plate mail	3	bronze plate mail	4
splint mail	4	banded mail	4
dwarvish mithril-coat	4	elven mithril-coat	5
chain mail	5	orcish chain mail	6
scale mail	6	dragon scales	7
studded leather armor	7	ring mail	7
orcish ring mail	8	leather armor	8
leather jacket	9	no armor	10

You can also wear other pieces of armor (cloak over suit, shirt under suit, helmet, gloves, boots, shield) to lower your armor class even further. Most of these provide a one or two point improvement to AC (making the overall value smaller and eventually negative) but can also be enchanted. Shirts are an exception; they don't provide any protection unless enchanted. Some cloaks also don't improve AC when unenchanted but all cloaks offer some protection against rust or corrosion to suits worn under them and against some monster *touch* attacks.

If a piece of armor is enchanted, its armor protection will be better (or worse) than normal, and its "plus" (or minus) will subtract from your armor class. For example, a +1 chain mail would give you better protection than normal chain mail, lowering your armor class one unit further to 4. When you put on a piece of armor, you immediately find out the armor class and any "plusses" it provides. Cursed pieces of armor usually have negative enchantments (minuses) in addition to being unremovable.

Many types of armor are subject to some kind of damage like rust. Such damage can be repaired. Some types of armor may inhibit spell casting.

The *nudist* option can be set (prior to game start) to attempt to play the entire game without wearing any armor (a self-imposed challenge which is extremely difficult to accomplish).

Some interfaces support the `armorstatus` option. When it is enabled, an extra status condition is displayed, summarizing currently worn armor.

The commands to use armor are '`W`' (wear) and '`T`' (take off). The '`A`' command can be used to take off armor as well as other worn items. Also, '`P`' (put on) and '`R`' (remove) which are normally for accessories can be used for armor, but pieces of armor won't be shown as likely candidates in a prompt for choosing what to put on or remove.

Food ('%')

Food is necessary to survive. If you go too long without eating you will faint, and eventually die of starvation. Some types of food will spoil, and become unhealthy to eat, if not protected. Food stored in ice boxes or tins ("cans") will usually stay fresh, but ice boxes are heavy, and tins take a while to open.

When you kill monsters, they usually leave corpses which are also "food." Many, but not all, of these are edible; some also give you special powers when you eat them. A good rule of thumb is "you are what you eat."

Some character roles and some monsters are vegetarian. Vegetarian monsters will typically never eat animal corpses, while vegetarian players can, but with some rather unpleasant side-effects.

You can name one food item after something you like to eat with the *fruit* option.

The command to eat food is '`e`'.

Scrolls ('?')

Scrolls are labeled with various titles, probably chosen by ancient wizards for their amusement value (for example, "READ ME," or "THANX MAUD" backwards). Scrolls disappear after you read them (except for blank ones, without magic spells on them).

One of the most useful of these is the *scroll of identify*, which can be used to determine what another object is, whether it is cursed or blessed, and how many uses it has left. Some objects of subtle enchantment are difficult to identify without these.

A scroll whose label is known can be read even when the hero is blind. If a scroll has been discovered, it will be listed in inventory by type rather than by label, but the label is known in that situation even though it isn't shown.

Many scrolls produce a different effect from usual if they are blessed or cursed, or read while the hero is confused.

A mail daemon may run up and deliver mail to you as a *scroll of mail* (on versions compiled with this feature). To use this feature on versions where *NetHack* mail delivery is triggered by electronic mail appearing in your system mailbox, you must let *NetHack* know where to look for new mail by setting the `MAIL` environment variable to the file name of your mailbox. You may also want to set the `MAILREADER` environment variable to the file name of your favorite reader, so *NetHack* can shell to it when you read the scroll. On versions of *NetHack* where mail is randomly generated internal to the game, these environment variables are ignored. You can disable the mail daemon by turning off the *mail* option.

The command to read a scroll is '`r`'.

Potions ('!')

Potions are distinguished by the color of the liquid inside the flask. They disappear after you quaff them.

Clear potions are potions of water. Sometimes these are blessed or cursed, resulting in holy or unholy water. Holy water is the bane of the undead, so potions of holy water are good things to throw ('`t`') at them. It is also sometimes very useful to dip ("`#dip`") an object into a potion.

The command to drink a potion is '`q`' (quaff).

Wands ('/')

Wands usually have multiple magical charges. Some types of wands require a direction in which to zap them. You can also zap them at yourself (just give a '`.`' or '`s`' for the direction). Be warned, however, for this is often unwise. Other types of wands don't require a direction. The number of charges in a wand is random and decreases by one whenever you use it.

When the number of charges left in a wand becomes zero, attempts to use the wand will usually result in nothing happening. Occasionally, however, it may be possible to squeeze the last few mana points from an otherwise spent wand, destroying it in the process. A wand may be recharged by using suitable magic, but doing so runs the risk of causing it to explode. The chance for such an explosion starts out very small and increases each time the wand is recharged.

In a truly desperate situation, when your back is up against the wall, you might decide to go for broke and break your wand. This is not for the faint of heart. Doing so will almost certainly cause a catastrophic release of magical energies.

When you have fully identified a particular wand, inventory display will include additional information in parentheses: the number of times it has been recharged followed by a colon and then by its current number of charges. A current charge count of `-1` is a special case indicating that the wand has been cancelled.

The command to use a wand is '`z`' (zap). To break one, use the '`a`' (apply) command.

Rings ('=')

Rings are very useful items, since they are relatively permanent magic, unlike the usually fleeting effects of potions, scrolls, and wands.

Putting on a ring activates its magic. You can wear at most two rings at any time, one on the ring finger of each hand.

Most worn rings also cause you to grow hungry more rapidly, the rate varying with the type of ring.

When wearing gloves, rings are worn underneath. If the gloves are cursed, rings cannot be put on and any already being worn cannot be removed. When worn gloves aren't cursed, you don't have to manually

take them off before putting on or removing a ring and then re-wear them after. That's done implicitly to avoid unnecessary tedium.

The commands to use rings are '`P`' (put on) and '`R`' (remove). '`A`', '`W`', and '`T`' can also be used; see *Amulets*.

Spellbooks ('+')

Spellbooks are tomes of mighty magic. When studied with the '`r`' (read) command, they transfer to the reader the knowledge of a spell (and therefore eventually become unreadable)—unless the attempt backfires. Reading a cursed spellbook or one with mystic runes beyond your ken can be harmful to your health!

A spell (even when learned) can also backfire when you cast it. If you attempt to cast a spell well above your experience level, or if you have little skill with the appropriate spell type, or cast it at a time when your luck is particularly bad, you can end up wasting both the energy and the time required in casting.

Casting a spell calls forth magical energies and focuses them with your naked mind. Some of the magical energy released comes from within you. Casting temporarily drains your magical power, which will slowly be recovered, and causes you to need additional food. Casting of spells also requires practice. With practice, your skill in each category of spell casting will improve. Over time, however, your memory of each spell will dim, and you will need to relearn it.

Some spells require a direction in which to cast them, similar to wands. To cast one at yourself, just give a '`.`' or '`s`' for the direction. A few spells require you to pick a target location rather than just specify a particular direction. Other spells don't require any direction or target.

Just as weapons are divided into groups in which a character can become proficient (to varying degrees), spells are similarly grouped. Successfully casting a spell exercises its skill group; using the "`#enhance`" command to advance a sufficiently exercised skill will affect all spells within the group. Advanced skill may increase the potency of spells, reduce their risk of failure during casting attempts, and improve the accuracy of the estimate for how much longer they will be retained in your memory. Skill slots are shared with weapons skills. (See also the section on "Weapon proficiency".)

Casting a spell also requires flexible movement, and wearing various types of armor may interfere with that.

The command to read a spellbook is the same as for scrolls, '`r`' (read). The '`+`' command lists each spell you know along with its level, skill category, chance of failure when casting, and an estimate of how strongly it is remembered. The '`Z`' (cast) command casts a spell.

Tools ('(')

Tools are miscellaneous objects with various purposes. Some tools have a limited number of uses, akin to wand charges. For example, lamps burn out after a while. Other tools are containers, which objects can be placed into or taken out of.

Some tools (such as a blindfold) can be *worn* and can be put on and removed like other accessories (rings, amulets); see *Amulets*. Other tools (such as pick-axe) can be wielded as weapons in addition to being applied for their usual purpose, and in some cases (again, pick-axe) become wielded as a weapon even when applied.

The *blind* option can be set (prior to game start) to attempt to play the entire game without being able to see (a self-imposed challenge which is very difficult to accomplish).

The command to use a tool is '`a`' (apply).

Containers

You may encounter bags, boxes, and chests in your travels. A tool of this sort can be opened with the "`#loot`" extended command when you are standing on top of it (that is, on the same floor spot), or with

the 'a' (apply) command when you are carrying it. However, chests are often locked, and are in any case unwieldy objects. You must set one down before unlocking it by using a key or lock-picking tool with the 'a' (apply) command, by kicking it with the '^D' command, or by using a weapon to force the lock with the "#force" extended command.

Some chests are trapped, causing nasty things to happen when you unlock or open them. You can check for and try to deactivate traps with the "#untrap" extended command.

When the contents of a container are known, that container will be described as something like "a sack containing 3 items". In this example, the 3 refers to number of *stacks* of compatible items, not to the total number of individual items. So a sack holding 2 sky blue potions, 7 arrows, and 350 gold pieces would be described as having 3 items rather than 10 or 359. And you would need to have 3 unused inventory slots available in order to take everything out (for the case where the items you remove don't combine into bigger stacks with things you're already carrying).

If a chest or large box is described as "broken", that means that it can't be locked rather than that it no longer functions as a container.

The *apply* and *loot* commands allow you to take out and/or put in an arbitrary number of items in a single operation. If you want to take everything out of a container, you can use the "#tip" command to pour the contents onto the floor. This may be your only way to get things out if your hands are stuck to a cursed two-handed weapon. When your hands aren't stuck, you have the potential to pour the contents into another container. (As of this writing, the other container must be carried rather than on the floor.)

Amulets ('"')

Amulets are very similar to rings, and often more powerful. Like rings, amulets have various magical properties, some beneficial, some harmful, which are activated by putting them on.

Only one amulet may be worn at a time, around your neck. Like wearing rings, wearing an amulet affects your metabolism, causing you to grow hungry more rapidly.

The commands to use amulets are the same as for rings, 'P' (put on) and 'R' (remove). 'A' can be used to remove various worn items including amulets. Also, 'W' (wear) and 'T' (take off) which are normally for armor can be used for amulets and other accessories (rings and eyewear), but accessories won't be shown as likely candidates in a prompt for choosing what to wear or take off.

Gems ('*')

Some gems are valuable, and can be sold for a lot of gold. They are also a far more efficient way of carrying your riches. Valuable gems increase your score if you bring them with you when you exit.

Other small rocks are also categorized as gems, but they are much less valuable. All rocks, however, can be used as projectile weapons (if you have a sling). In the most desperate of cases, you can still throw them by hand.

Large rocks ('`')

Statues and boulders are not particularly useful, and are generally heavy. It is rumored that some statues are not what they seem.

Boulders occasionally block your path. You can push one forward (by attempting to walk onto its spot) when nothing blocks *its* path, or you can smash it into a pile of small rocks with breaking magic or a pick-axe. It is possible to move onto a boulder's location if certain conditions are met; ordinarily one of those conditions is that pushing it any further be blocked. Using the move-without-picking-up prefix (default key 'm') prior to the direction of movement will attempt to move to a boulder's location without pushing it in addition to the prefix's usual action of suppressing auto-pickup at the destination.

Very large humanoids (giants and their ilk) have been known to pick up boulders and use them as missile weapons.

Unlike boulders, statues can't be pushed, but don't need to be because they don't block movement. They can be smashed into rocks though.

For some configurations of the program, statues are no longer shown as '`' but by the letter representing the monster they depict instead.

Gold ('$')

Gold adds to your score, and you can buy things in shops with it. There are a number of monsters in the dungeon that may be influenced by the amount of gold you are carrying (shopkeepers aside).

Gold pieces are the only type of object where bless/curse state does not apply. They're always uncursed but never described as uncursed even if you turn off the "*implicit_uncursed*" option. You can set the "*goldX*" option if you prefer to have gold pieces be treated as bless/curse state *unknown* rather than as known to be uncursed. Only matters when you're using an object selection prompt that can filter by "`BUCX`" state.

Persistence of Objects

Normally, if you have seen an object at a particular map location and move to another location where you can't directly see that object any more, it will continue to be displayed on your map. That remains the case even if it is not actually there any more—perhaps a monster has picked it up or it has rotted away—until you can see or feel that location again. One notable exception is that if the object gets covered by the "remembered, unseen monster" marker. When that marker is later removed after you've verified that no monster is there, you will have forgotten that there was any object there regardless of whether the unseen monster actually took the object. If the object is still there, then once you see or feel that location again you will re-discover the object and resume remembering it.

The situation is the same for a pile of objects, except that only the top item of the pile is displayed. The *hilite_pile* option can be enabled in order to show an item differently when it is the top one of a pile.

8 Conduct

As if winning *NetHack* were not difficult enough, certain players seek to challenge themselves by imposing restrictions on the way they play the game. The game automatically tracks some of these challenges, which can be checked at any time with the `#conduct` command or at the end of the game. When you perform an action which breaks a challenge, it will no longer be listed. This gives players extra "bragging rights" for winning the game with these challenges. Note that it is perfectly acceptable to win the game without resorting to these restrictions and that it is unusual for players to adhere to challenges the first time they win the game.

Several of the challenges are related to eating behavior. The most difficult of these is the foodless challenge. Although creatures can survive long periods of time without food, there is a physiological need for water; thus there is no restriction on drinking beverages, even if they provide some minor food benefits. Calling upon your god for help with starvation does not violate any food challenges either.

A strict vegan diet is one which avoids any food derived from animals. The primary source of nutrition is fruits and vegetables. The corpses and tins of blobs ('`b`'), jellies ('`j`'), and fungi ('`F`') are also considered to be vegetable matter. Certain human food is prepared without animal products; namely, lembas wafers, cram rations, food rations (gunyoki), K-rations, and C-rations. Metal or another normally indigestible material eaten while polymorphed into a creature that can digest it is also considered vegan food. Note however that eating such items still counts against foodless conduct.

Vegetarians do not eat animals; however, they are less selective about eating animal byproducts than vegans. In addition to the vegan items listed above, they may eat any kind of pudding ('`P`') other than the black puddings, eggs and food made from eggs (fortune cookies and pancakes), food made with milk (cream pies and candy bars), and lumps of royal jelly. Monks are expected to observe a vegetarian diet.

Eating any kind of meat violates the vegetarian, vegan, and foodless conducts. This includes tripe rations, the corpses or tins of any monsters not mentioned above, and the various other chunks of meat found in the dungeon. Swallowing and digesting a monster while polymorphed is treated as if you ate the creature's corpse. Eating leather, dragon hide, or bone items while polymorphed into a creature that can digest it, or eating monster brains while polymorphed into a mind flayer, is considered eating an animal, although wax is only an animal byproduct.

Regardless of conduct, there will be some items which are indigestible, and others which are hazardous to eat. Using a swallow-and-digest attack against a monster is equivalent to eating the monster's corpse. Please note that the term "vegan" is used here only in the context of diet. You are still free to choose not to use or wear items derived from animals (e.g. leather, dragon hide, bone, horns, coral), but the game will not keep track of this for you. Also note that "milky" potions may be a translucent white, but they do not contain milk, so they are compatible with a vegan diet. Slime molds or player-defined "fruits", although they could be anything from "cherries" to "pork chops", are also assumed to be vegan.

An atheist is one who rejects religion. This means that you cannot `#pray`, `#offer` sacrifices to any god, `#turn` undead, or `#chat` with a priest. Particularly selective readers may argue that playing Monk or Priest characters should violate this conduct; that is a choice left to the player. Offering the Amulet of Yendor to your god is necessary to win the game and is not counted against this conduct. You are also not penalized for being spoken to by an angry god, priest(ess), or other religious figure; a true atheist would hear the words but attach no special meaning to them.

A pauper starts the game with no possessions, no spells, and no weapon or spell skills (and if playing as a knight, your pony will not have a saddle). Can only be initiated by starting a new game with `OPTIONS=pauper` set in your run-time configurtion file or `NETHACKOPTIONS` environment variable. Once the game is underway, you can acquire and use items, spells, and skills in the usual way.

Most players fight with a wielded weapon (or tool intended to be wielded as a weapon). Another challenge is to win the game without using such a wielded weapon. You are still permitted to throw, fire, and kick weapons; use a wand, spell, or other type of item; or fight with your hands and feet.

In *NetHack*, a pacifist refuses to cause the death of any other monster (i.e. if you would get experience for the death). This is a particularly difficult challenge, although it is still possible to gain experience by other means.

An illiterate character does not read or write. This includes reading a scroll, spellbook, fortune cookie message, or t-shirt; writing a scroll; or making an engraving of anything other than a single "X" (the traditional signature of an illiterate person). Reading an engraving, or any item that is absolutely necessary to win the game, is not counted against this conduct. The identity of scrolls and spellbooks (and knowledge of spells) in your starting inventory is assumed to be learned from your teachers prior to the start of the game and isn't counted.

There is a side-branch to the main dungeon called "Sokoban," briefly described in the earlier section about *Traps*. As mentioned there, the goal is to push boulders into pits and/or holes to plug those in order to both get the boulders out of your way and be able to go past the traps. There are some special "rules" that are active when in that branch of the dungeon. Some rules can't be bypassed, such as being unable to push a boulder diagonally. Other rules can, such as not smashing boulders with magic or tools, but doing so causes you to receive a luck penalty. No message about that is given at the time, but it is tracked as a conduct. The `#conduct` command and end of game disclosure will report whether you have abided by the special rules of Sokoban, and if not, how many times you violated them, providing you with a way to discover which actions incur bad luck so that you can be better informed about whether or not to avoid repeating those actions in the future. (Note: the Sokoban conduct will only be displayed if you have entered the Sokoban branch of the dungeon during the current game. Once that has happened, it becomes part of disclosed conduct even if you haven't done anything interesting there. Ending the game with "never broke the Sokoban rules" conduct is most meaningful if you also manage to perform the "obtained the Sokoban prize" achievement (see *Achievements* below).)

There are several other challenges tracked by the game. It is possible to eliminate one or more species of monsters by genocide; playing without this feature is considered a challenge. When the game offers you an opportunity to genocide monsters, you may respond with the monster type "none" if you want

to decline. You can change the form of an item into another item of the same type ("polypiling") or the form of your own body into another creature ("polyself") by wand, spell, or potion of polymorph; avoiding these effects are each considered challenges. Polymorphing monsters, including pets, does not break either of these challenges. Finally, you may sometimes receive wishes; a game without an attempt to wish for any items is a challenge, as is a game without wishing for an artifact (even if the artifact immediately disappears). When the game offers you an opportunity to make a wish for an item, you may choose "nothing" if you want to decline.

Achievements

End of game disclosure will also display various achievements representing progress toward ultimate ascension, if any have been attained. They aren't directly related to *conduct* but are grouped with it because they fall into the same category of "bragging rights" and to limit the number of questions during disclosure. Listed here roughly in order of difficulty and not necessarily in the order in which you might accomplish them.

`<Rank>`	Attained rank title *Rank*.
`Shop`	Entered a shop.
`Temple`	Entered a temple.
`Mines`	Entered the Gnomish Mines.
`Town`	Entered Mine Town.
`Oracle`	Consulted the Oracle of Delphi.
`Novel`	Read a passage from a Discworld Novel.
`Sokoban`	Entered Sokoban.
`"Big Room"`	Entered the Big Room.
`"Soko-Prize"`	Explored to the top of Sokoban and found a special item there.
`Mines' End`	Explored to the bottom of the Gnomish Mines and found a special item there.
`Medusa`	Defeated Medusa.
`Tune`	Discovered the tune that can be used to open and close the drawbridge on the Castle level.
`Bell`	Acquired the Bell of Opening.
`Gehennom`	Entered Gehennom.
`Candle`	Acquired the Candelabrum of Invocation.
`Book`	Acquired the Book of the Dead.
`Invocation`	Gained access to the bottommost level of Gehennom.
`Amulet`	Acquired the fabled Amulet of Yendor.
`Endgame`	Reached the Elemental Planes.
`Astral`	Reached the Astral Plane level.
`Blind`	Blind from birth.
`Deaf`	Deaf from birth.
`Nudist`	Never wore any armor.
`Pauper`	Started out with no possessions.
`Ascended`	Delivered the Amulet to its final destination.

Notes:

Achievements are recorded and subsequently reported in the order in which they happen during your current game rather than the order listed here.

There are nine *Rank* titles for each role, bestowed at experience levels 1, 3, 6, 10, 14, 18, 22, 26, and 30. The one for experience level 1 is not recorded as an achievement. Losing enough levels to revert to lower rank(s) does not discard the corresponding achievement(s).

There's no guaranteed *Novel* so the achievement to read one might not always be attainable (except perhaps by wishing). Similarly, the *Big Room* level is not always present. Unlike with the Novel, there's no way to wish for this opportunity.

The "special items" hidden in *Mines' End* and *Sokoban* are not unique but are considered to be prizes or rewards for exploring those levels since doing so is not necessary to complete the game. Finding other

instances of the same objects doesn't record the corresponding achievement.

The *Medusa* achievement is recorded if she dies for any reason, even if you are not directly responsible, and only if she dies.

The 5-note *tune* can be learned via trial and error with a musical instrument played closely enough—but not too close!—to the Castle level's drawbridge or can be given to you via prayer boon.

Blind, *Deaf*, *Nudist*, and *Pauper* are also conducts, and they can only be enabled by setting the correspondingly named option in `NETHACKOPTIONS` or run-time configuration file prior to game start. In the case of *Blind* and *Deaf*, the option also enforces the conduct. They aren't really significant accomplishments unless/until you make substantial progress into the dungeon.

9 Options

Due to variations in personal tastes and conceptions of how *NetHack* should do things, there are options you can set to change how *NetHack* behaves.

Setting the options

Options may be set in a number of ways. Within the game, the '`O`' command allows you to view all options and change most of them. You can also set options automatically by placing them in a configuration file, or in the "`NETHACKOPTIONS`" environment variable. Some versions of *NetHack* also have front-end programs that allow you to set options before starting the game or a global configuration for system administrators.

Using a configuration file

The default name and location of the configuration file varies on different operating systems.

On UNIX, Linux and macOS it is ".nethackrc" in the user's home directory. The file may not exist, but it is a normal ASCII text file and can be created with any text editor.

On Windows, the name is ".nethackrc" location in the folder "%USERPROFILE%\NetHack\". The file may not exist, but it is a normal ASCII text file and can be created with any text editor. After running *NetHack* for the first time, you should find a default template for ths configuration file named ".nethackrc.template" in "%USERPROFILE%\NetHack\". If you have not created the configuration file, *NetHack* will create the configuration file for you using the default template file.

On MS-DOS it is "defaults.nh" in the same folder as *nethack.exe*.

Any line in the configuration file starting with '`#`' is treated as a comment and ignored. Empty lines are ignored.

Any line beginning with '`[`' and ending in '`]`' is a section marker (the closing '`]`' can be followed by whitespace and then an arbitrary comment beginning with '`#`'). The text between the square brackets is the section name. Section markers are only valid after a CHOOSE directive and their names are case-insensitive. Lines after a section marker belong to that section up until another section starts or a marker without a name is encountered or the file ends. Lines within sections are ignored unless a CHOOSE directive has selected that section.

You can use different configuration directives in the file, some of which can be used multiple times. In general, the directives are written in capital letters, followed by an equals sign, followed by settings particular to that directive.

Here is a list of allowed directives:

OPTIONS There are two types of options, boolean and compound options. Boolean options toggle a setting on or off, while compound options take more diverse values. Prefix a boolean option with 'no' or '!' to turn it off. For compound options, the option name and value are separated by a colon. Some options are persistent, and apply only to new games. You can specify multiple OPTIONS directives, and multiple options separated by commas in a single OPTIONS directive. (Comma separated options are processed from right to left.)
Example:

```
OPTIONS=dogname:Fido
OPTIONS=!legacy,autopickup,pickup_types:$"=/!?+
```

HACKDIR Default location of files *NetHack* needs. On Windows HACKDIR defaults to the location of the *NetHack.exe* or *NetHackw.exe* file so setting HACKDIR to override that is not usually necessary or recommended.

LEVELDIR The location that in-progress level files are stored. Defaults to HACKDIR, must be writable.

SAVEDIR The location where saved games are kept. Defaults to HACKDIR, must be writable.

BONESDIR The location that bones files are kept. Defaults to HACKDIR, must be writable.

LOCKDIR The location that file synchronization locks are stored. Defaults to HACKDIR, must be writable.

TROUBLEDIR The location that a record of game aborts and self-diagnosed game problems is kept. Defaults to HACKDIR, must be writable.

AUTOCOMPLETE Enable or disable an extended command autocompletion. Autocompletion has no effect for the X11 windowport. You can specify multiple autocompletions. To enable autocompletion, list the extended command. Prefix the command with "!" to disable the autocompletion for that command.
Example:

```
AUTOCOMPLETE=zap,!annotate
```

AUTOPICKUP_EXCEPTION Set exceptions to the *pickup_types* option. See the "Configuring Autopickup Exceptions" section.

BINDINGS Change the key bindings of some special keys, menu accelerators, extended commands, or mouse buttons. You can specify multiple bindings. Format is key followed by the command, separated by a colon. See the "Changing Key Bindings" section for more information.
Example:

```
BIND=^X:getpos.autodescribe
```

CHOOSE Chooses at random one of the comma-separated parameters as an active section name. Lines in other sections are ignored.
Example:

```
OPTIONS=color
CHOOSE=char A,char B
[char A]
OPTIONS=role:arc,race:dwa,align:law,gender:fem
[char B]
OPTIONS=role:wiz,race:elf,align:cha,gender:mal
[] #end of CHOOSE
OPTIONS=!rest_on_space
```

If `[]` is present, the preceding section is closed and no new section begins; whatever follows will be common to all sections. Otherwise the last section extends to the end of the options file.

MENUCOLOR Highlight menu lines with different colors. See the "Configuring Menu Colors" section.

MSGTYPE Change the way messages are shown in the top line. See the "Configuring Message Types" section.

ROGUESYMBOLS Custom symbols for the rogue level's symbol set. See *SYMBOLS* below.

SOUND Define a sound mapping. See the "Configuring User Sounds" section.

SOUNDDIR Define the directory that contains the sound files. See the "Configuring User Sounds" section.

SYMBOLS Override one or more symbols in the symbol set used for all dungeon levels except for the special rogue level. See the "Modifying *NetHack* Symbols" section.
Example:

```
# replace small punctuation (tick marks) with digits
SYMBOLS=S_golem:7
```

WIZKIT Debug mode only: extra items to add to initial inventory. Value is the name of a text file containing a list of item names, one per line, up to a maximum of 128 lines. Each line is processed by the function that handles wishing. Entries are added to the wish history; see the wizwish-command.
Example:

```
WIZKIT=~/wizkit.txt
```

Here is an example of configuration file contents:

```
# Set your character's role, race, gender, and alignment.
OPTIONS=role:Valkyrie, race:Human, gender:female, align:lawful

# Turn on autopickup, set automatically picked up object types
OPTIONS=autopickup,pickup_types:$"=/!?+

# Map customization
OPTIONS=color           # Display things in color if possible
OPTIONS=lit_corridor    # Show lit corridors differently
OPTIONS=hilite_pet,hilite_pile
# Replace small punctuation (tick marks) with digits
OPTIONS=boulder:0
SYMBOLS=S_golem:7

# No startup splash screen. Windows GUI only.
OPTIONS=!splash_screen
```

Using the NETHACKOPTIONS environment variable

The NETHACKOPTIONS variable is a comma-separated list of initial values for the various options. Some can only be turned on or off. You turn one of these on by adding the name of the option to the list, and turn it off by typing a '`!`' or "`no`" before the name. Others take a character string as a value. You can set string options by typing the option name, a colon or equals sign, and then the value of the string. The value is terminated by the next comma or the end of string.

For example, to set up an environment variable so that *color* is `on`, *legacy* is `off`, character *name* is set to "`Blue Meanie`", and named *fruit* is set to "`lime`", you would enter the command

```
setenv NETHACKOPTIONS "color,\!leg,name:Blue Meanie,fruit:lime"
```

in *csh* (note the need to escape the '!' since it's special to that shell), or the pair of commands

```
NETHACKOPTIONS="color,!leg,name:Blue Meanie,fruit:lime"
export NETHACKOPTIONS
```

in *sh*, *ksh*, or *bash*.

The NETHACKOPTIONS value is effectively the same as a single OPTIONS directive in a configuration file. The "OPTIONS=" prefix is implied and comma separated options are processed from right to left. Other types of configuration directives such as BIND or MSGTYPE are not allowed.

Instead of a comma-separated list of options, NETHACKOPTIONS can be set to the full name of a configuration file you want to use. If that full name doesn't start with a slash, precede it with '`@`' (at-sign) to let NetHack know that the rest is intended as a file name. If it does start with '`/`', the at-sign is optional.

Customization options

Here are explanations of what the various options do. Character strings that are too long may be truncated. Some of the options listed may be inactive in your dungeon.

Some options are persistent, and are saved and reloaded along with the game. Changing a persistent option in the configuration file applies only to new games.

accessiblemsg — Add location or direction information to messages (default is off).

acoustics — Enable messages about what your character hears (default on). Note that this has nothing to do with your computer's audio capabilities. Persistent.

alignment — Your starting alignment (`align:lawful`, `align:neutral`, or `align:chaotic`). You may specify just the first letter. Many roles and the non-human races restrict which alignments are allowed. See *role* for a description of how to use negation to exclude choices.
If `align` is not specified, there is no default value; player will be prompted unless role and/or race forces a choice for alignment. Cannot be set with the '`O`' command. Persistent.

armorstatus — Display an extra status condition which summarizes currently worn armor (default off, not supported by all interfaces).
For the usual case where more than one piece of armor is worn, a list of letters is shown in the following order:
`G` - gloves;
`C` - cloak;
`A` - suit;
`U` - shirt;
`H` - helmet;
`B` - boots;
`S` - shield.

autodescribe — Automatically describe the terrain under cursor when asked to get a location on the map (default true). The *whatis_coord* option controls whether the description includes map coordinates.

autodig — Automatically dig if you are wielding a digging tool and moving into a place that can be dug (default false). Persistent.

autoopen — Walking into a closed door attempts to open it (default true). Persistent.

autopickup — Automatically pick up things onto which you move (default off). Persistent.
See "*pickup_types*" and also "*autopickup_exception*" for ways to refine the behavior.
Note: prior to version 5.0.0, the default for *autopickup* was *on*.

autoquiver This option controls what happens when you attempt the '`f`' (fire) command when nothing is quivered or readied (default false). When true, the computer will fill your quiver or quiver sack or make ready some suitable weapon. Note that it will not take into account the blessed/cursed status, enchantment, damage, or quality of the weapon; you are free to manually fill your quiver or quiver sack or make ready with the '`Q`' command instead. If no weapon is found or the option is false, the '`t`' (throw) command is executed instead. Persistent.

autounlock Controls what action to take when attempting to walk into a locked door or to loot a locked container. Takes a plus-sign separated list of values:

`Untrap` prompt about whether to attempt to find a trap; it might fail to find one even when present; if it does find one, it will ask whether you want to try to disarm the trap; if you decline, your character will forget that the door or box is trapped;

`Apply-Key` if carrying a key or other unlocking tool, prompt about using it;

`Kick` kick the door (if you omit untrap or decline to attempt untrap and you omit apply-key or you lack a key or you decline to use the key; has no effect on containers);

`Force` try to force a container's lid with your currently wielded weapon (if you omit untrap or decline to attempt untrap and you omit apply-key or you lack a key or you decline to use the key; has no effect on doors);

`None` none of the above; can't be combined with the other choices.

Omitting the value is treated as if `autounlock:apply-key`. Preceding `autounlock` with '`!`' or "`no`" is treated as `autounlock:none`.
Applying a key might set off a trap if the door or container is trapped. Successfully kicking a door will break it and wake up nearby monsters. Successfully forcing a container open will break its lock and might also destroy some of its contents or damage your weapon or both.
The default is Apply-Key. Persistent.

blind Start the character permanently blind (default false). Persistent.

bones Allow saving and loading bones files (default true). Persistent.

boulder Set the character used to display boulders (default is the "large rock" class symbol, '`'`').

catname Name your starting cat (for example, "`catname:Morris`"). Cannot be set with the '`O`' command.

character Synonym for "`role`" to pick the type of your character (for example "`character:Monk`"). See *role* for more details.

checkpoint Save game state after each level change, for possible recovery after program crash (default on). Persistent.

cmdassist Have the game provide some additional command assistance for new players if it detects some anticipated mistakes (default on).

confirm Have user confirm attacks on pets, shopkeepers, and other peaceable creatures (default on). Persistent.

dark_room Show out-of-sight areas of lit rooms (default on). Persistent.

deaf Start the character permanently deaf (default false). Persistent.

dropped_nopick If this option is on, items you dropped will not be automatically picked up, even if "*autopickup*" is also on and they are in "*pickup_types*" or match a positive autopickup exception (default on). Persistent.

disclose Controls what information the program reveals when the game ends. Value is a space

separated list of prompting/category pairs (default is '`ni na nv ng nc no`', prompt with default response of '`n`' for each candidate). Persistent. The possibilities are:
`i` — disclose your inventory;
`a` — disclose your attributes;
`v` — summarize monsters that have been vanquished;
`g` — list monster species that have been genocided;
`c` — display your conduct; also achievements, if any;
`o` — display dungeon overview.
Each disclosure possibility can optionally be preceded by a prefix which lets you refine how it behaves. Here are the valid prefixes:
`y` — prompt you and default to yes on the prompt;
`n` — prompt you and default to no on the prompt;
`+` — disclose it without prompting;
`-` — do not disclose it and do not prompt.
The listing of vanquished monsters can be sorted, so there are two additional choices for '`v`': The listings of vanquished monsters and of genocided types can be sorted, so there are two additional choices for 'q' and 'g': `?` — prompt you and default to ask on the prompt;
`#` — disclose it without prompting, ask for sort order.
Asking refers to picking one of the orderings from a menu. The '`+`' disclose without prompting choice, or being prompted and answering '`y`' rather than '`a`', will default to showing monsters in the order specified by the *sortvanquished* option.
Omitted categories are implicitly added with '`n`' prefix. Specified categories with omitted prefix implicitly use '`+`' prefix. Order of the disclosure categories does not matter, program display for end-of-game disclosure follows a set sequence.
(for example, "`disclose:yi na +v -g o`") The example sets `inventory` to *prompt* and default to *yes*, `attributes` to *prompt* and default to *no*, `vanquished` to *disclose without prompting*, `genocided` to *not disclose* and *not prompt*, `conduct` to implicitly *prompt* and default to *no*, `overview` to *disclose without prompting*.
Note that the vanquished monsters list includes all monsters killed by traps and each other as well as by you. And the dungeon overview shows all levels you had visited but does not reveal things about them that you hadn't discovered.

dogname Name your starting dog (for example, "`dogname:Fang`"). Cannot be set with the '`O`' command.

extmenu Changes the extended commands interface to pop-up a menu of available commands. It is keystroke compatible with the traditional interface except that it does not require that you hit Enter. It is implemented for the tty interface (default off). .lp "" For the X11 interface, which always uses a menu for choosing an extended command, it controls whether the menu shows all available commands (on) or just the subset of commands which have traditionally been considered extended ones (off).

female An obsolete synonym for "`gender:female`". Cannot be set with the '`O`' command.

fireassist This option controls what happens when you attempt the '`f`' (fire) and don't have an appropriate launcher, such as a bow or a sling, wielded. If on, you will automatically wield the launcher. Default is on.

fixinv An object's inventory letter sticks to it when it's dropped (default on). If this is off, dropping an object shifts all the remaining inventory letters. Persistent.

force_invmenu Commands asking for an inventory item show a menu instead of a text query with possible menu letters. Default is off.

fruit Name a fruit after something you enjoy eating (for example, "`fruit:mango`") (default "`slime mold`"). Basically a nostalgic whimsy that *NetHack* uses from time to time.

You should set this to something you find more appetizing than slime mold. Apples, oranges, pears, bananas, and melons already exist in *NetHack*, so don't use those.

gender Your starting gender (`gender:male` or `gender:female`). You may specify just the first letter. Although you can still denote your gender using either of the deprecated "*male*" and "*female*" options, the "*gender*" option will take precedence. See *role* for a description of how to use negation to exclude choices.
If `gender` is not specified, there is no default value; player will be prompted unless role and/or race forces a choice for gender. Cannot be set with the '`O`' command. Persistent.

goldX When filtering objects based on bless/curse state (BUCX), whether to treat gold pieces as `X` (unknown bless/curse state, when 'on') or `U` (known to be uncursed, when 'off', the default). Gold is never blessed or cursed, but it is not described as "uncursed" even when the *implicit_uncursed* option is 'off'.

help If more information is available for an object looked at with the '`/`' command, ask if you want to see it (default on). Turning help off makes just looking at things faster, since you aren't interrupted with the "`More info?`" prompt, but it also means that you might miss some interesting and/or important information. Persistent.

herecmd_menu When using a windowport that supports mouse and clicking on yourself or next to you, show a menu of possible actions for the location. Same as "`#herecmdmenu`" and "`#therecmdmenu`" commands.

hilite_pet Visually distinguish pets from similar animals (default off). The behavior of this option depends on the type of windowing you use. In text windowing, text highlighting or inverse video is often used; with tiles, generally displays a heart symbol near pets. With the tty or curses interface, the *petattr* option controls how to highlight pets and setting it will turn the *hilite_pet* option on or off as warranted.

hilite_pile Visually distinguish piles of objects from individual objects (default off). The behavior of this option depends on the type of windowing you use. In text windowing, text highlighting or inverse video is often used; with tiles, generally displays a small plus-symbol beside the object on the top of the pile.

hitpointbar Show a hit point bar graph behind your name and title in the status display (default off).
The "curses" interface supports it even if the status highlighting feature has been disabled when building the program. The "tty" and "mswin" (aka "Windows GUI") interfaces support it only if status highlighting is left enabled when building. You don't need to set up any highlighting rules in order to display the bar. If there is one for hitpoints in effect and it specifies color, that color will be used for the bar. However if it specifies video attributes, they will be ignored in favor of *inverse*. For tty and curses, *blink* will also be used if the current hitpoint value is at or below the *critical HP* threshold.
The "Qt" interface also supports hitpointbar, by drawing a solid bar above the name and title with a hard-coded color scheme. (As of this writing, having the bar enabled unintentionally inhibits resizing the status panel. To resize that, use the `#optionsfull` command to toggle the *hitpointbar* option off, perform the resize while it's off, then use the same command to toggle it back on.)

horsename Name your starting horse (for example, "`horsename:Trigger`"). Cannot be set with the '`O`' command.

ignintr Ignore interrupt signals, including breaks (default off). Persistent.

implicit_uncursed Omit "uncursed" from object descriptions when it can be deduced from other aspects of the description (default on). Persistent.

If you use menu coloring, you may want to turn this off.

legacy Display an introductory message when starting the game (default on). Persistent.

lit_corridor Show corridor squares seen by night vision or a light source held by your character as lit (default off). Persistent.

lootabc When using a menu to interact with a container, use the old '`a`', '`b`', and '`c`' keyboard shortcuts rather than the mnemonics '`o`', '`i`', and '`b`' (default off). Persistent.

mail Enable mail delivery during the game (default on). Persistent.

male An obsolete synonym for "`gender:male`". Cannot be set with the '`O`' command.

mention_decor Give feedback when walking onto various dungeon features such as stairs, fountains, or altars which are ordinarily only described when covered by one or more objects (default off). Persistent.

mention_map Give feedback when interesting map locations change (default off).

mention_walls Give feedback when walking against a wall (default off). Persistent.

menucolors Enable coloring menu lines (default off). See "*Configuring Menu Colors*" on how to configure the colors.

menustyle Controls the method used when you need to choose various objects (in response to the `Drop` (aka `droptype`) command, for instance). The value specified should be the first letter of one of the following: traditional, combination, full, or partial. Default is `full`. Persistent.
Traditional was the only method available for very early versions; it consists of a prompt for object class characters, followed by an object-by-object prompt for all items matching the selected object class(es). Combination starts with a prompt for object class(es) of interest, but then displays a menu of matching objects rather than prompting one-by-one. Full displays a menu of object classes rather than a character prompt, and then a menu of matching objects for selection. (Choosing its 'A' (Autoselect-All) choice skips the second menu. To avoid choosing that by accident, set *paranoid_confirm:AutoAll* to require confirmation.) Partial skips the object class filtering and immediately displays a menu of all objects.

menu_deselect_all Key to deselect all items in a menu. Implemented by the Amiga, Gem, X11 and tty ports. Default '-'.

menu_deselect_page Key to deselect all items on this page of a menu. Implemented by the Amiga, Gem and tty ports. Default '\'.

menu_first_page Key to jump to the first page in a menu. Implemented by the Amiga, Gem and tty ports. Default '^'.

menu_headings Controls how the headings in a menu are highlighted. Takes a text attribute, or text color and attribute separated by ampersand. For allowed attributes and colors, see "*Configuring Menu Colors*". Not all ports can actually display all types.

menu_invert_all Key to invert all items in a menu. Implemented by the Amiga, Gem, X11 and tty ports. Default '@'.

menu_invert_page Key to invert all items on this page of a menu. Implemented by the Amiga, Gem and tty ports. Default '~'.

menu_last_page Key to jump to the last page in a menu. Implemented by the Amiga, Gem and tty ports. Default '—'.

menu_next_page Key to go to the next menu page. Implemented by the Amiga, Gem and tty ports. Default '¿'.

menu_objsyms Inventory and other object menus are normally separated by object class (weapons, armor, and so forth), with a menu header line at the beginning of each group. You

can have menus add the display symbol for the class of objects for each header line. You can also add the display symbol for the individual item in each menu entry. For a tiles map, that would be a small rendition of an object's tile. For a text map, it is the same character as is used for the object's class, which would be most useful when there are no headers separating objects among classes.
Supported by tty and curses. When setting the value, it can be specified by digit or keyword. The default value is `Conditional` (4).

menu_overlay Do not clear the screen before drawing menus, and align menus to the right edge of the screen. Only for the tty port. (default on)

menu_previous_page Key to go to the previous menu page. Implemented by the Amiga, Gem and tty ports. Default '¡'.

menu_search Key to search for some text and toggle selection state of matching menu items. Default ':'.

menu_select_all Key to select all items in a menu. Implemented by the Amiga, Gem, X11 and tty ports. Default '.'.

menu_select_page Key to select all items on this page of a menu. Implemented by the Amiga, Gem and tty ports. Default ','.

menu_shift_left Key to scroll a menu—one which has been scrolled right—back to the left. Implemented for *perm_invent* only by curses and X11. Default '{'.

menu_shift_right Key to scroll a menu which has text beyond the right edge to the right. Implemented for *perm_invent* only by curses by X11. Default '}'.

mon_movement Show a message when hero notices a monster movement (default is off).

monpolycontrol Prompt for new form whenever any monster changes shape (default off). Debug mode only.

montelecontrol Prompt for destination whenever any monster gets teleported (default off). Debug mode only.

mouse_support Allow use of the mouse for input and travel. Valid settings are:
`0` — disabled
`1` — enabled and make OS adjustments to support mouse use
`2` — like `1`, but does not make any OS adjustments

Omitting a value is the same as specifying `1` and negating *mouse_support* is the same as specifying `0`.

msghistory The number of top line messages to save (and be able to recall with '`^P`') (default 20). Cannot be set with the '`O`' command.

msg_window Allows you to change the way recalled messages are displayed. Currently it is only supported for tty (all four choices) and for curses ('`f`' and '`r`' choices, default '`r`'). The possible values are:
`s` — single message (default; only choice prior to 3.4.0);
`c` — combination, two messages as *single*, then as *full*;
`f` — full window, oldest message first;
`r` — full window reversed, newest message first.
For backward compatibility, no value needs to be specified (which defaults to *full*), or it can be negated (which defaults to *single*).

name Set your character's name (defaults to your user name). You can also set your character's role by appending a dash and one or more letters of the role (that is, by suffixing one of "`-A -B -C -H -K -M -P -Ra -Ro -S -T -V -W`"). If "`-@`" is used for the role, then a random one will be automatically chosen.
On some systems, the default is the player's user name; on others, there is no default

and the player will be prompted. The former can made to behave like the latter by specifying a generic name such as "player". Cannot be set with the '`O`' command.

news Read the *NetHack* news file, if present (default on). Since the news is shown at the beginning of the game, there's no point in setting this with the '`O`' command.

nudist Start the character with no armor (default false). Persistent.

null Send padding nulls to the terminal (default on). Persistent.

number_pad Use digit keys instead of letters to move (default 0 or off).
Valid settings are:
`0` — move by letters; '`yuhjklbn`'
`1` — move by numbers; digit '`5`' acts as '`G`' movement prefix
`2` — like `1` but '`5`' works as '`g`' prefix instead of as '`G`'
`3` — by numbers using phone key layout; `123` above, `789` below
`4` — combines `3` with `2`; phone layout plus MS-DOS compatibility
`-1` — by letters but use '`z`' to go northwest, '`y`' to zap wands
For backward compatibility, omitting a value is the same as specifying `1` and negating *number_pad* is the same as specifying `0`. (Settings `2` and `4` are for compatibility with MS-DOS or old PC Hack; in addition to the different behavior for '`5`', '`Alt-5`' acts as '`G`' and '`Alt-O`' acts as '`I`'. Setting `-1` is to accommodate some QWERTZ keyboards which have the location of the '`y`' and '`z`' keys swapped.) When moving by numbers, to enter a count prefix for those commands which accept one (such as "`12s`" to search twelve times), precede it with the letter '`n`' ("`n12s`").

packorder Specify the order to list object types in (default "`")[%?+!=/(*`0_`"). The value of this option should be a string containing the symbols for the various object types. Any omitted types are filled in at the end from the previous order.

paranoid_confirmation A space separated list of specific situations where alternate prompting is desired. The default is "*paranoid_confirmation:pray swim trap*".

`Confirm` for any prompts which are set to require "yes" rather than 'y', also require "no" to reject instead of accepting any non-yes response as no; changes pray and AutoAll to require "yes" or "no" too;

`quit` require "`yes`" rather than '`y`' to confirm quitting the game or switching into non-scoring explore mode;

`die` require "`yes`" rather than '`y`' to confirm dying (not useful in normal play; applies to explore mode);

`bones` require "`yes`" rather than '`y`' to confirm saving bones data when dying in debug mode

`attack` require "`yes`" rather than '`y`' to confirm attacking a peaceful monster;

`wand-break` require "`yes`" rather than '`y`' to confirm breaking a wand with the *apply* command;

`eating` require "`yes`" rather than '`y`' to confirm whether to continue eating;

`Were-change` require "`yes`" rather than '`y`' to confirm changing form due to lycanthropy when hero has polymorph control;

`pray` require '`y`' to confirm an attempt to pray rather than immediately praying; on by default; (to require "yes" rather than just 'y', set Confirm too);

`trap` require '`y`' to confirm an attempt to move into or onto a known trap, unless doing so is considered to be harmless; when enabled, this confirmation is also used for moving into visible gas cloud regions; (to require "yes" rather than just 'y', set Confirm too); confirmation can be skipped by using the '`m`' movement prefix;

`swim` prevent walking into water or lava; on by default; (to deliberately step

onto/into such terrain when this is set, use the 'm' movement prefix when adjacent);

`AutoAll` require confirmation when the 'A' (Autoselect-All) choice is selected in object class filtering menus for *menustyle:Full*; (to require "yes" rather than just 'y', set Confirm too);

`Remove` require selection from inventory for 'R' and 'T' commands even when wearing just one applicable item;

`all` turn on all of the above.

By default, the pray, swim, and trap choices are enabled, the others disabled. To disable them without setting any of the other choices, use "*paranoid_confirmation:none*". To keep them enabled while setting any of the others, you can include them in the list, such as "*paranoid_confirmation:attack pray swim Remove*" or you can precede the first entry in the list with a plus sign, "*paranoid_confirmation:+attack Remove*". To remove an entry that has been previously set without removing others, precede the first entry in the list with a minus sign, "*paranoid_confirmation:-swim*. To both add some new entries and remove some old ones, you can use multiple *paranoid_confirmation* option settings, or you can use the '+' form and list entries to be added by their name and entries to be removed by '!' and name. The positive (no '!') and negative (with '!') entries can be intermixed.

pauper Start the character with no possessions (default false). Persistent.
Also start with no spells or skills, which are tied to starting equipment. Does not inhibit acquiring and using items, spells, and skills once play has started.

perm_invent If true, always display your current inventory in a window (default is false).
This only makes sense for windowing system interfaces that implement this feature. For those that do, the `perminv_mode` option can be used to refine what gets displayed for *perm_invent*. Setting that to a value other than *none* while *perm_invent* is false will change it to true.

perminv_mode Augments the `perm_invent` option. Value is one of

`none` behave as if *perm_invent* is false;
all show all inventory except for gold;
full show full inventory including gold;
in-use only show items which are in use (worn, wielded, lit lamp).

Default is *none* but if *perm_invent* gets set to true while it is *none* it will be changed to *all*.
Note: if gold has been equipped in quiver/ammo-pouch then it will be included for *all* despite that mode normally omitting gold.

petattr Specifies one or more text highlighting attributes to use when showing pets on the map. Effectively a superset of the *hilite_pet* boolean option. Curses or tty interface only; value is one of none, bold, dim, underline, italic, blink, and inverse. Some of those choices might not work, depending upon terminal hardware or terminal emulation software.

pettype Specify the type of your initial pet, if you are playing a character class that uses multiple types of pets; or choose to have no initial pet at all. Possible values are "`cat`", "`dog`", "`horse`" and "`none`". If the choice is not allowed for the role you are currently playing, it will be silently ignored. For example, "`horse`" will only be honored when playing a knight. Cannot be set with the 'O' command.

pickup_burden When you pick up an item that would exceed this encumbrance level (Unencumbered, Burdened, streSsed, straiNed, overTaxed, or overLoaded), you will be asked if you want to continue. (Default 'S'). Persistent.

pickup_stolen If this option is on and "*autopickup*" is also on, try to pick up things that a monster stole from you, even if they aren't in "*pickup_types*" or match an autopickup exception. Default is on. Persistent.

pickup_thrown If this option is on and "*autopickup*" is also on, try to pick up things that you threw, even if they aren't in "*pickup_types*" or match an autopickup exception. Default is on. Persistent.

pickup_types Specify the object types to be picked up when "*autopickup*" is on. Default is all types. Persistent.
The value is a list of object symbols, such as `pickup_types:$?!&` to pick up gold, scrolls, and potions. You can use "*autopickup_exception*" configuration file lines to further refine "*autopickup*" behavior.
There is no way to set *pickup_types* to "*none*". (Setting it to an empty value reverts to "*all*".) If you want to avoid automatically picking up any types of items but do want to have *autopickup* on in order to have *autopickup_exceptions* control what you do and don't pick up, you can set *pickup_types* to '`.`'. That is the type symbol for *venom* and you won't come across any venom items so won't unintentionally pick such up.

pile_limit When walking across a pile of objects on the floor, threshold at which the message "there are few/several/many objects here" is given instead of showing a popup list of those objects. A value of 0 means "no limit" (always list the objects); a value of 1 effectively means "never show the objects" since the pile size will always be at least that big; default value is 5. Persistent.

playmode Values are *normal*, *explore*, or *debug*. Allows selection of explore mode (also known as discovery mode) or debug mode (also known as wizard mode) instead of normal play. Debug mode might only be allowed for someone logged in under a particular user name (on multi-user systems) or specifying a particular character name (on single-user systems) or it might be disabled entirely. Requesting it when not allowed or not possible results in explore mode instead. Default is normal play.

price_quotes Whenever the game mentions the name of an object you haven't identified yet, it also mentions the range of buy and sell prices you have seen for that item (to help narrow down what it could be). The price shown is the unit price for one item (even when you are looking at a stack of multiple items). Many players may want to turn this on while identifying objects, and then turn it back off again for general play. Default is off.

pushweapon Using the '`w`' (wield) command when already wielding something pushes the old item into your alternate weapon slot (default off). Likewise for the '`a`' (apply) command if it causes the applied item to become wielded. Persistent.

query_menu Use a menu when asked specific yes/no queries, instead of a prompt.

quick_farsight When set, usually prevents the "you sense your surroundings" message where play pauses to allow you to browse the map whenever clairvoyance randomly activates. Some situations, such as being underwater or engulfed, ignore this option. It does not affect the clairvoyance spell where pausing to examine revealed objects or monsters is less intrusive. Default is off. Persistent.

race Choices are `human`, `dwarf`, `elf`, `gnome`, and `orc` but most roles restrict which of the non-human races are allowed. See *role* for a description of how to use negation to exclude choices.
If `race` is not specified, there is no default value; player will be prompted unless role forces a choice for race. Cannot be set with the '`O`' command. Persistent.

reroll Allows rerolling your character's starting inventory and attributes (default false).

Persistent.
Note that rerolling your character is not a recommended way to play if aiming merely to win (a lucky start has a much smaller influence on whether or not you win the game than your actions later in the game). This option exists partly as an acknowledgement that some players will insist on doing so anyway, and partly because rerolling may be necessary for certain types of challenge games.

rest_on_space Make the space bar a synonym for the '`.`' (#wait) command (default off). Persistent.

role Pick your type of character (for example, "`role:Samurai`"); synonym for "*character*". See "*name*" for an alternate method of specifying your role. This option can also be used to limit selection when role is chosen randomly. Use a space-separated list of roles and either negate each one or negate the option itself instead. Negation is accomplished in the same manner as with *boolean options*, by prefixing the option or its value(s) with '`!`' or "`no`".
Examples:

```
OPTIONS=role:!arc !bar !kni
OPTIONS=!role:arc bar kni
```

There can be multiple instances of the *role* option if they're all negations.
If `role` is not specified, there is no default value; player will be prompted. Cannot be set with the '`O`' command. Persistent.

roguesymset This option may be used to select one of the named symbol sets found within `symbols` to alter the symbols displayed on the screen on the rogue level.

rlecomp When writing out a save file, perform run length compression of the map. Not all ports support run length compression. It has no effect on reading an existing save file.

runmode Controls the amount of screen updating for the map window when engaged in multi-turn movement (running via `shift`+direction or `control`+direction and so forth, or via the travel command or mouse click). The possible values are:
`teleport` — update the map after movement has finished;
`run` — update the map after every seven or so steps;
`walk` — update the map after each step;
`crawl` — like *walk*, but pause briefly after each step.
This option only affects the game's screen display, not the actual results of moving. The default is *run*; versions prior to 3.4.1 used *teleport* only. Whether or not the effect is noticeable will depend upon the window port used or on the type of terminal. Persistent.

safe_pet Prevent you from (knowingly) attacking your pets (default on). Persistent.

safe_wait Prevents you from waiting or searching when next to a hostile monster (default on). Persistent.

sanity_check Evaluate monsters, objects, and map prior to each turn (default off). Debug mode only.

scores Control what parts of the score list you are shown at the end (for example, "`scores:5top scores/4around my score/own scores`"). Only the first letter of each category ('`t`', '`a`' or '`o`') is necessary. Persistent.

showdamage Whenever your character takes damage, show a message of the damage taken, and the amount of hit points left.

showexp Show your accumulated experience points on bottom line (default off). Persistent.

showrace Display yourself as the glyph for your race, rather than the glyph for your role (default

off). Note that this setting affects only the appearance of the display, not the way the game treats you. Persistent.

showscore Show your approximate accumulated score on bottom line (default off). By default, this feature is suppressed when building the program. Persistent.

showvers Include the game's version number on the status lines (default off). Potentially useful if you switch between different versions or variants, or you are making screenshots or streaming video. Using the *statuslines:3* option is recommended so that there will be more room available for status information, unless you're using NetHack's *Qt* interface or your terminal emulator window displays fewer than 25 lines. Persistent. The *versinfo* option provides limited control over what information *showvers* displays.

silent Suppress terminal beeps (default on). Persistent.

sortdiscoveries Controls the sorting behavior for the output of the '\' and '`' commands. Persistent.
The possible values are:
`o` — list object types by class, in discovery order within each class; default;
`s` — list object types by *sortloot* classification: by class, by sub-class within class for classes which have substantial groupings (like helmets, boots, gloves, and so forth for armor), with object types partly-discovered via assigned name coming before fully identified types;
`c` — list by class, alphabetically within each class;
`a` — list alphabetically across all classes.
Can be interactively set via the '`O`' command or via using the '`m`' prefix before the '\' or '`' command.

sortloot Controls the sorting behavior of pickup lists for inventory and #loot commands and some others. Persistent.
The possible values are:
`full` — always sort the lists;
`loot` — only sort the lists that don't use inventory letters, like with the #loot and pickup commands;
`none` — show lists the traditional way without sorting; default.

sortpack Sort the pack contents by type when displaying inventory (default on). Persistent.

sortvanquished Controls the sorting behavior for the output of the `#vanquished` command and also for the `#genocided` command. Persistent.
The possible values are:
`t` — traditional: order by monster level; ties are broken by internal monster index; default;
`d` — order by monster difficulty rating; ties broken by internal index;
`a` — order alphabetically, first any unique monsters then all the others;
`c` — order by monster class, by low to high level within each class;
`n` — order by count, high to low; ties are broken by internal monster index;
`z` — order by count, low to high; ties broken by internal index.
Can be interactively set via the '`m O`' command or via using the '`m`' prefix before either the `#vanquished` command or the `#genocided` command.

sounds Allow sounds to be emitted from an integrated sound library (default on).

sparkle Display a sparkly effect when a monster (including yourself) is hit by an attack to which it is resistant (default on). Persistent.

spot_monsters Show a message when hero notices a monster (default is off).

standout Boldface monsters and "`--More--`" (default off). Persistent.

statushilites Controls how many turns status hilite behaviors highlight the field. If negated or set to zero, disables status hiliting. See "*Configuring Status Hilites*" for further information.

status_updates Allow updates to the status lines at the bottom of the screen (default true).

suppress_alert This option may be set to a *NetHack* version level to suppress alert notification messages about feature changes for that and prior versions (for example, "`suppress_alert:3.3.1`")

symset This option may be used to select one of the named symbol sets found within `symbols` to alter the symbols displayed on the screen. Use "`symset:default`" to explicitly select the default symbols.

terrainstatus Display an extra status condition describing the spot beneath the hero's feet (default off, not supported by all interfaces).

time Show the elapsed game time in turns on bottom line (default off). Persistent.

timed_delay When pausing momentarily for display effect, such as with explosions and moving objects, use a timer rather than sending extra characters to the screen. (Applies to "tty" and "curses" interfaces only; "X11" interface always uses a timer-based delay. The default is on if configured into the program.) Persistent.

tips Show some helpful tips during gameplay (default on). Persistent.

tombstone Draw a tombstone graphic upon your death (default on). Persistent.

toptenwin Put the ending display in a *NetHack* window instead of on stdout (default off). Setting this option makes the score list visible when a windowing version of *NetHack* is started without a parent window, but it no longer leaves the score list around after game end on a terminal or emulating window.

travel Allow the travel command via mouse click (default on). Turning this option off will prevent the game from attempting unintended moves if you make inadvertent mouse clicks on the map window. Does not affect traveling via the '_' ("`#travel`") command. Persistent.

tutorial Play a tutorial level at the start of the game. Setting this option on or off in the config file will skip the query.

verbose Provide more commentary during the game (default on). Persistent.

versinfo Controls what the *showvers* option displays on the status lines.

weaponstatus Display an extra status condition which describes currently wielded weapon (default off, not supported by all interfaces).
Some possible displayed values are:
`bare-hnds` - no weapon and no gloves;
`empty-hnd` - no weapon but gloves are worn;
`dual-weps` - wielding two weapons.

whatis_coord When using the '`/`' or '`;`' commands to look around on the map with "`autodescribe`" on, display coordinates after the description. Also works in other situations where you are asked to pick a location.

The possible settings are:
`c` — `compass ('east' or '3s' or '2n,4w');`
`f` — `full compass ('east' or '3south' or '2north,4west');`
`m` — `map <x,y> (map column x=0 is not used);`
`s` — `screen [row,column] (row is offset to match tty usage);`
`n` — `none (no coordinates shown) [default].`
The *whatis_coord* option is also used with the '`/m`', '`/M`', '`/o`', and '`/O`' sub-commands of '`/`', where the '*none*' setting is overridden with '*map*'.

whatis_filter When getting a location on the map, and using the keys to cycle through next and previous targets, allows filtering the possible targets. (default none)
The possible settings are:

`n` — `no filtering`;
`v` — `in view only`;
`a` — `in same area (room, corridor, etc)`. The area-filter tries to be slightly predictive—if you're standing on a doorway, it will consider the area on the side of the door you were last moving towards.
Filtering can also be changed when getting a location with the "getpos.filter" key.

whatis_menu When getting a location on the map, and using a key to cycle through next and previous targets, use a menu instead to pick a target. (default off)

whatis_moveskip When getting a location on the map, and using shifted movement keys or meta-digit keys to fast-move, instead of moving 8 units at a time, move by skipping the same glyphs. (default off)

windowtype When the program has been built to support multiple interfaces, select whichone to use, such as "`tty`" or "`X11`" (default depends on build-time settings; use "`#version`" to check). Cannot be set with the '`O`' command.
When used, it should be the first option set since its value might enable or disable the availability of various other options. For multiple lines in a configuration file, that would be the first non-comment line. For a comma-separated list in NETHACKOPTIONS or an OPTIONS line in a configuration file, that would be the *rightmost* option in the list.

wizweight Augment object descriptions with their objects' weight (default off). Debug mode only.

zerocomp When writing out a save file, perform zero-comp compression of the contents. Not all ports support zero-comp compression. It has no effect on reading an existing save file.

Window Port Customization options

Here are explanations of the various options that are used to customize and change the characteristics of the windowtype that you have chosen. Character strings that are too long may be truncated. Not all window ports will adjust for all settings listed here. You can safely add any of these options to your configuration file, and if the window port is capable of adjusting to suit your preferences, it will attempt to do so. If it can't it will silently ignore it. You can find out if an option is supported by the window port that you are currently using by checking to see if it shows up in the Options list. Some options are dynamic and can be specified during the game with the '`O`' command.

align_message Where to align or place the message window (top, bottom, left, or right)

align_status Where to align or place the status window (top, bottom, left, or right).

ascii_map If *NetHack* can, it should display the map using simple characters (letters and punctuation) rather than *tiles* graphics. In some cases, characters can be augmented with line-drawing symbols; use the `symset` option to select a symbol set such as *DECgraphics* or *IBMgraphics* if your display supports them. Setting `ascii_map` to *True* forces `tiled_map` to be *False*.

color If *NetHack* can, it should display color for different monsters, objects, and dungeon features (default on).

eight_bit_tty If *NetHack* can, it should pass eight-bit character values (for example, specified with the *traps* option) straight through to your terminal (default off).

font_map If *NetHack* can, it should use a font by the chosen name for the map window.

font_menu If *NetHack* can, it should use a font by the chosen name for menu windows.

font_message If *NetHack* can, it should use a font by the chosen name for the message window.

font_status If *NetHack* can, it should use a font by the chosen name for the status window.

font_text If *NetHack* can, it should use a font by the chosen name for text windows.

font_size_map If *NetHack* can, it should use this size font for the map window.

font_size_menu If *NetHack* can, it should use this size font for menu windows.

font_size_message If *NetHack* can, it should use this size font for the message window.

font_size_status If *NetHack* can, it should use this size font for the status window.

font_size_text If *NetHack* can, it should use this size font for text windows.

fullscreen If *NetHack* can, it should try to display on the entire screen rather than in a window.

guicolor Use color text and/or highlighting attributes when displaying some non-map data (such as menu selector letters). Curses interface only; default is on.

large_font If *NetHack* can, it should use a large font.

map_mode If *NetHack* can, it should display the map in the manner specified.

player_selection If *NetHack* can, it should pop up dialog boxes or use prompts for character selection.

popup_dialog If *NetHack* can, it should pop up dialog boxes for input.

preload_tiles If *NetHack* can, it should preload tiles into memory. For example, in the protected mode MS-DOS version, control whether tiles get pre-loaded into RAM at the start of the game. Doing so enhances performance of the tile graphics, but uses more memory. (default on). Cannot be set with the '`O`' command.

scroll_amount If *NetHack* can, it should scroll the display by this number of cells when the hero reaches the scroll_margin.

scroll_margin If *NetHack* can, it should scroll the display when the hero or cursor is this number of cells away from the edge of the window.

selectsaved If *NetHack* can, it should display a menu of existing saved games for the player to choose from at game startup, if it can. Not all ports support this option.

softkeyboard If *NetHack* can, it should display an onscreen keyboard. Handhelds are most likely to support this option.

splash_screen If *NetHack* can, it should display an opening splash screen when it starts up (default yes).

statuslines Number of lines for traditional below-the-map status display. Acceptable values are `2` and `3` (default is `2`).
When set to `3`, the `tty` interface moves some fields around and mainly shows status conditions on their own line. A display capable of showing at least 25 lines is recommended. The value can be toggled back and forth during the game with the '`O`' command.
The `curses` interface does likewise if the *align_status* option is set to *top* or *bottom* but ignores *statuslines* when set to *left* or *right*.
The `Qt` interface already displays more than 3 lines for status so uses the *statuslines* value differently. A value of `3` renders status in the `Qt` interface's original format, with the status window spread out vertically. A value of `2` makes status be slightly condensed, moving some fields to different lines to eliminate one whole line, reducing the height needed. (If NetHack has been built using a version of `Qt` older than `qt-5.9`, *statuslines* can only be set in the run-time configuration file or via NETHACKOPTIONS, not during play with the '`O`' command.)

term_cols and
term_rows Curses interface only. Number of columns and rows to use for the display. Curses will attempt to resize to the values specified but will settle for smaller sizes if they are too big. Default is the current window size.

tile_file Specify the name of an alternative tile file to override the default.
Note: the X11 interface uses X resources rather than NetHack's options to select an alternate tile file. See `NetHack.ad`, the sample X "application defaults" file.

tile_height Specify the preferred height of each tile in a tile capable port.

tile_width Specify the preferred width of each tile in a tile capable port

tiled_map If *NetHack* can, it should display the map using *tiles* graphics rather than simple characters (letters and punctuation, possibly augmented by line-drawing symbols). Setting `tiled_map` to *True* forces `ascii_map` to be *False*.

use_darkgray Use bold black instead of blue for black glyphs (TTY only).

use_inverse If *NetHack* can, it should display inverse when the game specifies it.

use_menu_glyphs If *NetHack* can, it should display glyphs next to objects in the inventory.

vary_msgcount If *NetHack* can, it should display this number of messages at a time in the message window.

windowborders Whether to draw boxes around the map, status area, message area, and persistent inventory window if enabled. Curses interface only. Acceptable values are
`0` — off, never show borders
`1` — on, always show borders
`2` — auto, on display is at least $(24 + 2) \times (80 + 2)$ [default]
`3` — on, except forced off for perm_invent
`4` — auto, except forced off for perm_invent

(The 26x82 size threshold for '2' refers to number of rows and columns of the display. A width of at least 110 columns $(80 + 2 + 26 + 2)$ is needed for *align_status* set to `left` or `right`.)
The persistent inventory window, when enabled, can grow until it is too big to fit on most displays, resulting in truncation of its contents. If borders are forced on (1) or the display is big enough to show them (2), setting the value to 3 or 4 instead will keep borders for the map, message, and status windows but have room for two additional lines of inventory plus widen each inventory line by two columns.

windowcolors If *NetHack* can, it should display all windows of a particular style with the specified foreground and background colors. Windows GUI and curses windowport only. The format is
`OPTION=windowcolors:`*style foreground*`/`*background*
where *style* is one of `menu`, `message`, `status`, or `text`, and *foreground* and *background* are colors, either numeric (hash sign followed by three pairs of hexadecimal digits, *#rrggbb*), one of the named colors (*black*, *red*, *green*, *brown*, *blue*, *magenta*, *cyan*, *orange*, *bright-green*, *yellow*, *bright-blue*, *bright-magenta*, *bright-cyan*, *white*, *gray*, *purple*, *silver*, *maroon*, *fuchsia*, *lime*, *olive*, *navy*, *teal*, *aqua*), or (for Windows only) one of Windows UI colors (*trueblack*, *activeborder*, *activecaption*, *appworkspace*, *background*, *btnface*, *btnshadow*, *btntext*, *captiontext*, *graytext*, *greytext*, *highlight*, *highlighttext*, *inactiveborder*, *inactivecaption*, *menu*, *menutext*, *scrollbar*, *window*, *windowframe*, *windowtext*).

wraptext If *NetHack* can, it should wrap long lines of text if they don't fit in the visible area of the window.

Crash Report Options

Please note that NetHack does not send **a**ny information off your computer unless you manually click submit on a form.

OPTION=crash_email:*email_address*

OPTION=crash_name:*your_name*

These options are used only to save you some typing on the crash report and #bugreport forms.

OPTION=crash_urlmax:*bytes*

This option is used to limit the length of the URLs generated and is only needed if your browser cannot handle arbitrarily long URLs.

Platform-specific Customization options

Here are explanations of options that are used by specific platforms or ports to customize and change the port behavior.

altkeyhandling Select an alternate way to handle keystrokes (*Win32 tty NetHack* only). The name of the handling type is one of *default*, *ray*, *340*

altmeta On systems where this option is available, it can be set to tell *NetHack* to convert a two character sequence beginning with ESC into a meta-shifted version of the second character (default off).
This conversion is only done for commands, not for other input prompts. Note that typing one or more digits as a count prefix prior to a command—preceded by `n` if the *number_pad* option is set—is also subject to this conversion, so attempting to abort the count by typing ESC will leave *NetHack* waiting for another character to complete the two character sequence. Type a second ESC to finish cancelling such a count. At other prompts a single ESC suffices.

BIOS Use BIOS calls to update the screen display quickly and to read the keyboard (allowing the use of arrow keys to move) on machines with an IBM PC compatible BIOS ROM (default off, *OS/2, PC* and *ST NetHack* only).

rawio Force raw (non-cbreak) mode for faster output and more bulletproof input (MS-DOS sometimes treats '`^P`' as a printer toggle without it) (default off, *OS/2, PC* and *ST NetHack* only). Note: DEC Rainbows hang if this is turned on. Cannot be set with the '`O`' command.

subkeyvalue (*Win32 tty NetHack* only). May be used to alter the value of keystrokes that the operating system returns to *NetHack* to help compensate for international keyboard issues. OPTIONS=subkeyvalue:171/92 will return 92 to *NetHack*, if 171 was originally going to be returned. You can use multiple subkeyvalue assignments in the configuration file if needed. Cannot be set with the '`O`' command.

video Set the video mode used (*PC NetHack* only). Values are *autodetect*, *default*, *vga*, or *vesa*. Setting *vesa* will cause the game to display tiles, using the full capability of the VGA hardware. Setting *vga* will cause the game to display tiles, fixed at 640x480 in 16 colors, a mode that is compatible with all VGA hardware. Third party tilesets will probably not work. Setting *autodetect* attempts *vesa*, then *vga*, and finally sets *default* if neither of those modes works. Cannot be set with the '`O`' command.

video_height Set the VGA mode resolution height (MS-DOS only, with video:vesa)

video_width Set the VGA mode resolution width (MS-DOS only, with video:vesa)

videocolors Set the color palette for PC systems using NO_TERMS (default 4-2-6-1-5-3-15-12-10-14-9-13-11, *PC NetHack* only). The order of colors is red, green, brown, blue, magenta, cyan, bright.white, bright.red, bright.green, yellow, bright.blue, bright.magenta, and bright.cyan. Cannot be set with the '`O`' command.

videoshades Set the intensity level of the three gray scales available (default dark normal light, *PC NetHack* only). If the game display is difficult to read, try adjusting these scales; if

this does not correct the problem, try `!color`. Cannot be set with the '`O`' command.

Regular Expressions

Regular expressions are normally POSIX extended regular expressions. It is possible to compile *NetHack* without regular expression support on a platform where there is no regular expression library. While this is not true of any modern platform, if your *NetHack* was built this way, patterns are instead glob patterns; regardless, this document refers to both as "regular expressions." This applies to Autopickup exceptions, Message types, Menu colors, and User sounds.

Configuring Autopickup Exceptions

You can further refine the behavior of the "`autopickup`" option beyond what is available through the "`pickup_types`" option.

By placing "`autopickup_exception`" lines in your configuration file, you can define patterns to be checked when the game is about to autopickup something.

autopickup_exception Sets an exception to the "*pickup_types*" option. The *autopickup_exception* option should be followed by a regular expression to be used as a pattern to match against the singular form of the description of an object at your location.
In addition, some characters are treated specially if they occur as the first character in the pattern, specifically:
`<` — always pickup an object that matches rest of pattern;
`>` — never pickup an object that matches rest of pattern.
The *autopickup_exception* rules are processed in the order in which they appear in your configuration file, thus allowing a later rule to override an earlier rule.
Exceptions can be set with the '`O`' command, but because they are not included in your configuration file, they won't be in effect if you save and then restore your game. *autopickup_exception* rules are not saved with the game.

Here are some examples:

```
autopickup_exception="<*arrow"
autopickup_exception=">*corpse"
autopickup_exception=">* cursed*"
```

The first example above will result in autopickup of any type of arrow. The second example results in the exclusion of any corpse from autopickup. The last example results in the exclusion of items known to be cursed from autopickup.

Changing Key Bindings

It is possible to change the default key bindings of some special commands, menu accelerator keys, extended commands, by using BIND stanzas in the configuration file. Format is key, followed by the command to bind to, separated by a colon. The key can be a single character ("`x`"), a control key ("`^X`", "`C-x`"), a meta key ("`M-x`"), a mouse button, or a three-digit decimal ASCII code.

For example:

```
BIND=^X:getpos.autodescribe
BIND=\:menu_first_page
BIND=v:loot
```

`Extended command keys` You can bind multiple keys to the same extended command. Unbind a key by using "`nothing`" as the extended command to bind to. You can also bind the "`<esc>`", "`<enter>`", and "`<space>`" keys.

`Menu accelerator keys` The menu control or accelerator keys can also be rebound via OPTIONS lines in the configuration file. You cannot bind object symbols or selection letters into menu accelerators. Some interfaces only support some of the menu accelerators.

`Mouse buttons` You can bind "mouse1" or "mouse2" to "`nothing`", "`therecmdmenu`", "`clicklook`", or "`mouseaction`".

`Special command keys` Below are the special commands you can rebind. Some of them can be bound to same keys with no problems, others are in the same "context", and if bound to same keys, only one of those commands will be available. Special command can only be bound to a single key.

count Prefix key to start a count, to repeat a command this many times. With *number_pad* only. Default is '`n`'.

getdir.help When asked for a direction, the key to show the help. Default is '`?`'.

getdir.mouse When asked for a direction, the key to initiate a simulated mouse click. You will be asked to pick a location. Use movement keystrokes to move the cursor around the map, then type the getpos.pick.once key (default '`,`') or the getpos.pick key (default '`.`') to finish as if performing a left or right click. Only useful when using the `#therecmdmenu` command. Default is '_'.

getdir.self When asked for a direction, the key to target yourself. Default is '`.`'.

getdir.self2 When asked for a direction, an alternate key to target yourself. Default is '`s`'.

getpos.autodescribe When asked for a location, the key to toggle *autodescribe*. Default is '`#`'.

getpos.all.next When asked for a location, the key to go to next closest interesting thing. Default is '`a`'.

getpos.all.prev When asked for a location, the key to go to previous closest interesting thing. Default is '`A`'.

getpos.door.next When asked for a location, the key to go to next closest door or doorway. Default is '`d`'.

getpos.door.prev When asked for a location, the key to go to previous closest door or doorway. Default is '`D`'.

getpos.help When asked for a location, the key to show help. Default is '`?`'.

getpos.mon.next When asked for a location, the key to go to next closest monster. Default is '`m`'.

getpos.mon.prev When asked for a location, the key to go to previous closest monster. Default is '`M`'.

getpos.obj.next When asked for a location, the key to go to next closest object. Default is '`o`'.

getpos.obj.prev When asked for a location, the key to go to previous closest object. Default is '`O`'.

getpos.menu When asked for a location, and using one of the next or previous keys to cycle through targets, toggle showing a menu instead. Default is '`!`'.

getpos.moveskip When asked for a location, and using the shifted movement keys or meta-digit keys to fast-move around, move by skipping the same glyphs instead of by 8 units. Default is '`*`'.

getpos.filter When asked for a location, change the filtering mode when using

one of the next or previous keys to cycle through targets. Toggles between no filtering, in view only, and in the same area only. Default is '"'.

getpos.pick When asked for a location, the key to choose the location, and possibly ask for more info. When simulating a mouse click after being asked for a direction (see getdir.mouse above), the key to use to respond as right click. Default is '.'.

getpos.pick.once When asked for a location, the key to choose the location, and skip asking for more info. When simulating a mouse click after being asked for a direction, the key to respond as left click. Default is ','.

getpos.pick.quick When asked for a location, the key to choose the location, skip asking for more info, and exit the location asking loop. Default is ';'.

getpos.pick.verbose When asked for a location, the key to choose the location, and show more info without asking. Default is ':'.

getpos.self When asked for a location, the key to go to your location. Default is '@'.

getpos.unexplored.next When asked for a location, the key to go to next closest unexplored location. Default is 'x'.

getpos.unexplored.prev When asked for a location, the key to go to previous closest unexplored location. Default is 'X'.

getpos.valid When asked for a location, the key to go to show valid target locations. Default is '$'.

getpos.valid.next When asked for a location, the key to go to next closest valid location. Default is 'z'.

getpos.valid.prev When asked for a location, the key to go to previous closest valid location. Default is 'Z'.

Configuring Message Types

You can change the way the messages are shown in the message area, when the message matches a user-defined pattern.

In general, the configuration file entries to describe the message types look like this:

```
MSGTYPE=type "pattern"
```

type how the message should be shown:
`show` — show message normally.
`hide` — never show the message.
`stop` — wait for user with more-prompt.
`norep` — show the message once, but not again if no other message is shown in between.

pattern the pattern to match. The pattern should be a regular expression.

Here's an example of message types using *NetHack's* internal pattern matching facility:

```
MSGTYPE=stop "You feel hungry."
MSGTYPE=hide "You displaced *."
```

specifies that whenever a message "You feel hungry" is shown, the user is prompted with more-prompt, and a message matching "You displaced ¡something¿" is not shown at all.

The order of the defined MSGTYPE lines is important; the last matching rule is used. Put the general case first, exceptions below them.

Configuring Menu Colors

Some platforms allow you to define colors used in menu lines when the line matches a user-defined pattern. At this time the tty, curses, win32tty and win32gui interfaces support this.

In general, the configuration file entries to describe the menu color mappings look like this:

```
MENUCOLOR="pattern"=color&attribute
```

pattern — the pattern to match;

color — the color to use for lines matching the pattern;

attribute — the attribute to use for lines matching the pattern. The attribute is optional, and if left out, you must also leave out the preceding ampersand. If no attribute is defined, no attribute is used.

The pattern should be a regular expression.

Allowed colors are *black*, *red*, *green*, *brown*, *blue*, *magenta*, *cyan*, *gray*, *orange*, *light-green*, *yellow*, *light-blue*, *light-magenta*, *light-cyan*, and *white*. And *no-color*, the default foreground color, which isn't necessarily the same as any of the other colors.

Allowed attributes are *none*, *bold*, *dim*, *italic*, *underline*,*blink*, and *inverse*. *Normal* is a synonym for *none*. Note that the platform used may interpret the attributes any way it wants.

Here's an example of menu colors using *NetHack's* internal pattern matching facility:

```
MENUCOLOR="* blessed *"=green
MENUCOLOR="* cursed *"=red
MENUCOLOR="* cursed *(being worn)"=red&underline
```

specifies that any menu line with " blessed " contained in it will be shown in green color, lines with " cursed " will be shown in red, and lines with " cursed " followed by "(being worn)" on the same line will be shown in red color and underlined. You can have multiple MENUCOLOR entries in your configuration file, and the last MENUCOLOR line that matches a menu line will be used for the line.

Note that if you intend to have one or more color specifications match " uncursed ", you will probably want to turn the *implicit_uncursed* option off so that all items known to be uncursed are actually displayed with the "uncursed" description.

Configuring User Sounds

Some platforms allow you to define sound files to be played when a message that matches a user-defined pattern is delivered to the message window. At this time the Qt port and the win32tty and win32gui ports support the use of user sounds.

The following configuration file entries are relevant to mapping user sounds to messages:

SOUNDDIR — The directory that houses the sound files to be played.

SOUND — An entry that maps a sound file to a user-specified message pattern. Each SOUND entry is broken down into the following parts:
`MESG` — message window mapping (the only one supported in 5.0.0);
`msgtype` — optional; message type to use, see "Configuring User Sounds"
`pattern` — the pattern to match;
`sound file` — the sound file to play;
`volume` — the volume to be set while playing the sound file;
`sound index` — optional; the index corresponding to a sound file.

The pattern should be a regular expression.

For example:

```
SOUNDDIR=C:\nethack\sounds
SOUND=MESG "This door is locked" "lock.wav" 100
SOUND=MESG hide "^You miss the " "swing.wav" 75
```

Configuring Status Hilites

Your copy of *NetHack* may have been compiled with support for *Status Hilites*. If so, you can customize your game display by setting thresholds to change the color or appearance of fields in the status display.

The format for defining status colors is:

```
OPTION=hilite_status:field-name/behavior/color&attributes
```

For example, the following line in your configuration file will cause the hitpoints field to display in the color red if your hitpoints drop to or below a threshold of 30

```
OPTION=hilite_status:hitpoints/<=30%/red/normal
```

(That example is actually specifying `red& normal` for `<=30%` and `no-color& normal` for `>30%`.)

For another example, the following line in your configuration file will cause wisdom to be displayed red if it drops and green if it rises:

```
OPTION=hilite_status:wisdom/down/red/up/green
```

Allowed colors are black, red, green, brown, blue, magenta, cyan, gray, orange, light-green, yellow, light-blue, light-magenta, light-cyan, and white. And *no-color*, the default foreground color on the display, which is not necessarily the same as black or white or any of the other colors.

Allowed attributes are none, bold, dim, underline, italic, blink, and inverse. "Normal" is a synonym for "none"; they should not be used in combination with any of the other attributes.

To specify both a color and an attribute, use '&' to combine them. To specify multiple attributes, use '+' to combine those.

For example: `magenta& inverse+dim`.

Note that the display may substitute or ignore particular attributes depending upon its capabilities, and in general may interpret the attributes any way it wants. For example, on some display systems a request for bold might yield blink or vice versa. On others, issuing an attribute request while another is already set up will replace the earlier attribute rather than combine with it. Since nethack issues attribute requests sequentially (at least with the *tty* interface) rather than all at once, the only way a situation like that can be controlled is to specify just one attribute.

You can adjust the display of the following status fields:

title	dungeon-level	experience-level
strength	gold	experience
dexterity	hitpoints	HD
constitution	hitpoints-max	time
intelligence	power	hunger
wisdom	power-max	carrying-capacity
charisma	armor-class	condition
alignment		score

The pseudo-field 'characteristics' can be used to set all six of Str, Dex, Con, Int, Wis, and Cha at once. 'HD' is 'hit dice', an approximation of experience level displayed when polymorphed. 'experience', 'time', and 'score' are conditionally displayed depending upon your other option settings.

Instead of a behavior, 'condition' takes the following condition flags: *stone*, *slime*, *strngl*, *foodpois*, *termill*, *blind*, *deaf*, *stun*, *conf*, *hallu*, *lev*, *fly*, and *ride*. You can use 'major_troubles' as an alias for stone

through termill, 'minor_troubles' for blind through hallu, 'movement' for lev, fly, and ride, and 'all' for every condition.

Allowed behaviors are "always", "up", "down", "changed", a percentage or absolute number threshold, or text to match against. For the *hitpoints* field, the additional behavior "criticalhp" is available. It overrides other behavior rules if hit points are at or below the *major problem* threshold (which varies depending upon maximum hit points and experience level).

`always` will set the default attributes for that field.

`up` , `down` set the field attributes for when the field value changes upwards or downwards. This attribute times out after `statushilites` turns.

`changed` sets the field attribute for when the field value changes. This attribute times out after `statushilites` turns. (If a field has both a "changed" rule and an "up" or "down" rule which matches a change in the field's value, the "up" or "down" one takes precedence.)

`percentage` sets the field attribute when the field value matches the percentage. It is specified as a number between 0 and 100, followed by '`%`' (percent sign). If the percentage is prefixed with '`<=`' or '`>=`', it also matches when value is below or above the percentage. Use prefix '`<`' or '`>`' to match when strictly below or above. (The numeric limit is relaxed slightly for those: `>-1%` and `<101%` are allowed.) Only four fields support percentage rules. Percentages for "*hitpoints*" and "*power*" are straightforward; they're based on the corresponding maximum field. Percentage highlight rules are also allowed for "*experience level*" and "*experience points*" (valid when the *showexp* option is enabled). For those, the percentage is based on the progress from the start of the current experience level to the start of the next level. So if level 2 starts at 20 points and level 3 starts at 40 points, having 30 points is 50% and 35 points is 75%. 100% is unattainable for experience because you'll gain a level and the calculations will be reset for that new level, but a rule for `=100%` is allowed and matches the special case of being exactly 1 experience point short of the next level.

`absolute` value sets the attribute when the field value matches that number. The number must be 0 or higher, except for "*armor-class* which allows negative values, and may optionally be preceded by '`=`'. If the number is preceded by '`<=`' or '`>=`' instead, it also matches when value is below or above. If the prefix is '`<`' or '`>`', only match when strictly above or below.

`criticalhp` only applies to the hitpoints field and only when current hit points are below a threshold (which varies by maximum hit points and experience level). When the threshold is met, a criticalhp rule takes precedence over all other hitpoints rules.

`text` match sets the attribute when the field value matches the text. Text matches can only be used for "*alignment*", "*carrying-capacity*", "*hunger*", "*dungeon-level*", and "*title*". For title, only the role's rank title is tested; the character's name is ignored.

The in-game options menu can help you determine the correct syntax for a configuration file.
The whole feature can be disable by setting option *statushilites* to 0.
Example hilites:

```
OPTION=hilite_status: gold/up/yellow/down/brown
OPTION=hilite_status: characteristics/up/green/down/red
OPTION=hilite_status: hitpoints/100%/gray&normal
OPTION=hilite_status: hitpoints/<100%/green&normal
OPTION=hilite_status: hitpoints/<66%/yellow&normal
OPTION=hilite_status: hitpoints/<50%/orange&normal
OPTION=hilite_status: hitpoints/<33%/red&bold
OPTION=hilite_status: hitpoints/<15%/red&inverse
```

```
OPTION=hilite_status: condition/major/orange&inverse
OPTION=hilite_status: condition/lev+fly/red&inverse
```

Modifying *NetHack* Symbols

NetHack can load entire symbol sets from the symbol file.

The options that are used to select a particular symbol set from the symbol file are:

symset Set the name of the symbol set that you want to load. *symbols*.

roguesymset Set the name of the symbol set that you want to load for display on the rogue level.

You can also override one or more symbols using the *SYMBOLS* and *ROGUESYMBOLS* configuration file options. Symbols are specified as *name:value* pairs. Note that *NetHack* escape-processes the *value* string in conventional C fashion. This means that '\' is a prefix to take the following character literally. Thus '\' needs to be represented as '\\'. The special prefix '\m' switches on the meta bit in the symbol value, and the '^' prefix causes the following character to be treated as a control character.

Table 1: NetHack Symbols

Default	Symbol Name	Description
@	S_air	(air)
_	S_altar	(altar)
"	S_amulet	(amulet)
A	S_angel	(angelic being)
a	S_ant	(ant or other insect)
^	S_anti_magic_trap	(anti-magic field)
[	S_armor	(suit or piece of armor)
[	S_armour	(suit or piece of armor)
^	S_arrow_trap	(arrow trap)
0	S_ball	(iron ball)
#	S_bars	(iron bars)
B	S_bat	(bat or bird)
^	S_bear_trap	(bear trap)
-	S_blcorn	(bottom left corner)
b	S_blob	(blob)
+	S_book	(spellbook)
)	S_boomleft	(boomerang open left)
(	S_boomright	(boomerang open right)
`	S_boulder	(boulder)
-	S_brcorn	(bottom right corner)
¿	S_brdnladder	(branch ladder down)
¿	S_brdnstair	(branch staircase down)
¡	S_brupladder	(branch ladder up)
¡	S_brupstair	(branch staircase up)
C	S_centaur	(centaur)
_	S_chain	(iron chain)
#	S_cloud	(cloud)
c	S_cockatrice	(cockatrice)
$	S_coin	(pile of coins)
#	S_corr	(corridor)
-	S_crwall	(wall)
-	S_darkroom	(dark room)
^	S_dart_trap	(dart trap)
&	S_demon	(major demon)
	S_digbeam	(dig beam)
¿	S_dnladder	(ladder down)

Table 1: NetHack Symbols

Default	Symbol Name	Description
¿	S_dnstair	(staircase down)
d	S_dog	(dog or other canine)
D	S_dragon	(dragon)
;	S_eel	(sea monster)
E	S_elemental	(elemental)
#	S_engrcorr	(engraving in a corridor)
`	S_engroom	(engraving in a room)
/	S_expl_tl	(explosion top left)
-	S_expl_tc	(explosion top center)
\	S_expl_tr	(explosion top right)
—	S_expl_ml	(explosion middle left)
	S_expl_mc	(explosion middle center)
—	S_expl_mr	(explosion middle right)
\	S_expl_bl	(explosion bottom left)
-	S_expl_bc	(explosion bottom center)
/	S_expl_br	(explosion bottom right)
e	S_eye	(eye or sphere)
^	S_falling_rock_trap	(falling rock trap)
f	S_feline	(cat or other feline)
^	S_fire_trap	(fire trap)
!	S_flashbeam	(flash beam)
%	S_food	(piece of food)
{	S_fountain	(fountain)
F	S_fungus	(fungus or mold)
	S_gem	(gem or rock)
	S_ghost	(ghost)
H	S_giant	(giant humanoid)
G	S_gnome	(gnome)
'	S_golem	(golem)
—	S_grave	(grave)
g	S_gremlin	(gremlin)
-	S_hbeam	(wall)
#	S_hcdbridge	(horizontal raised drawbridge)
+	S_hcdoor	(closed door)
.	S_hodbridge	(horizontal lowered drawbridge)
—	S_hodoor	(open door)
^	S_hole	(hole)
@	S_human	(human or elf)
h	S_humanoid	(humanoid)
-	S_hwall	(horizontal wall)
.	S_ice	(ice)
i	S_imp	(imp or minor demon)
I	S_invisible	(invisible monster)
J	S_jabberwock	(jabberwock)
j	S_jelly	(jelly)
k	S_kobold	(kobold)
K	S_kop	(Keystone Kop)
^	S_land_mine	(land mine)
}	S_lava	(molten lava)
}	S_lavawall	(wall of lava)
1	S_leprechaun	(leprechaun)
^	S_level_teleporter	(level teleporter)
L	S_lich	(lich)
y	S_light	(light)

Table 1: NetHack Symbols

Default	Symbol Name	Description
#	S_litcorr	(lit corridor)
:	S_lizard	(lizard)
\	S_lslant	(wall)
^	S_magic_portal	(magic portal)
^	S_magic_trap	(magic trap)
m	S_mimic	(mimic)
]	S_mimic_def	(mimic)
M	S_mummy	(mummy)
N	S_naga	(naga)
.	S_ndoor	(doorway)
n	S_nymph	(nymph)
O	S_ogre	(ogre)
o	S_orc	(orc)
p	S_piercer	(piercer)
^	S_pit	(pit)
#	S_poisoncloud	(poison cloud)
^	S_polymorph_trap	(polymorph trap)
}	S_pool	(water)
!	S_potion	(potion)
P	S_pudding	(pudding or ooze)
q	S_quadruped	(quadruped)
Q	S_quantmech	(quantum mechanic)
=	S_ring	(ring)
`	S_rock	(boulder or statue)
r	S_rodent	(rodent)
^	S_rolling_boulder_trap	(rolling boulder trap)
.	S_room	(floor of a room)
/	S_rslant	(wall)
^	S_rust_trap	(rust trap)
R	S_rustmonst	(rust monster or disenchanter)
?	S_scroll	(scroll)
#	S_sink	(sink)
^	S_sleeping_gas_trap	(sleeping gas trap)
S	S_snake	(snake)
s	S_spider	(arachnid or centipede)
^	S_spiked_pit	(spiked pit)
^	S_squeaky_board	(squeaky board)
0	S_ss1	(magic shield 1 of 4)
#	S_ss2	(magic shield 2 of 4)
@	S_ss3	(magic shield 3 of 4)
	S_ss4	(magic shield 4 of 4)
^	S_statue_trap	(statue trap)
	S_stone	(solid rock)
]	S_strange_obj	(strange object)
-	S_sw_bc	(swallow bottom center)
\	S_sw_bl	(swallow bottom left)
/	S_sw_br	(swallow bottom right)
—	S_sw_ml	(swallow middle left)
—	S_sw_mr	(swallow middle right)
-	S_sw_tc	(swallow top center)
/	S_sw_tl	(swallow top left)
\	S_sw_tr	(swallow top right)
-	S_tdwall	(wall)
^	S_teleportation_trap	(teleportation trap)

Table 1: NetHack Symbols

Default	Symbol Name	Description
\	S_throne	(opulent throne)
-	S_tlcorn	(top left corner)
—	S_tlwall	(wall)
(	S_tool	(useful item (pick-axe, key, lamp...))
^	S_trap_door	(trap door)
t	S_trapper	(trapper or lurker above)
-	S_trcorn	(top right corner)
#	S_tree	(tree)
T	S_troll	(troll)
—	S_trwall	(wall)
-	S_tuwall	(wall)
U	S_umber	(umber hulk)
	S_unexplored	(unexplored terrain)
u	S_unicorn	(unicorn or horse)
¡	S_upladder	(ladder up)
¡	S_upstair	(staircase up)
V	S_vampire	(vampire)
—	S_vbeam	(wall)
#	S_vcdbridge	(vertical raised drawbridge)
+	S_vcdoor	(closed door)
.	S_venom	(splash of venom)
^	S_vibrating_square	(vibrating square)
.	S_vodbridge	(vertical lowered drawbridge)
-	S_vodoor	(open door)
v	S_vortex	(vortex)
—	S_vwall	(vertical wall)
/	S_wand	(wand)
}	S_water	(water)
}	S_weapon	(weapon)
”	S_web	(web)
w	S_worm	(worm)
~	S_worm_tail	(long worm tail)
W	S_wraith	(wraith)
x	S_xan	(xan or other extraordinary insect)
X	S_xorn	(xorn)
Y	S_yeti	(apelike creature)
Z	S_zombie	(zombie)
z	S_zruty	(zruty)
	S_pet_override	(any pet if ACCESSIBILITY=1 is set)
@	S_hero_override	(hero if ACCESSIBILITY=1 is set)

Notes:

Several symbols in this table appear to be blank. They are the space character, except for S_pet_override and S_hero_override which don't have any default value and can only be used if enabled in the "sysconf" file.

S_rock is misleadingly named; rocks and stones use S_gem. Statues and boulders are the rock being referred to, but since version 3.6.0, statues are displayed as the monster they depict. So S_rock is only used for boulders and not used at all if overridden by the more specific S_boulder.

Customizing Map Glyph Representations Using Unicode

If your platform or terminal supports the display of UTF-8 character sequences, you can customize your game display by assigning Unicode codepoint values and red-green-blue colors to glyph representations. The customizations can be specified for use with a symset that has a UTF8 handler within the symbols file such as the enhanced1 set, or individually within your own nethack.rc file.

The format for defining a glyph representation is:

```
OPTIONS=glyph:glyphid/U+nnnn/R-G-B
```

The window port that is active needs to provide support for displaying UTF-8 character sequences and explicit 24-bit red-green-blue colors in order for the glyph representation to be visible as specified.

For example, the following line in your configuration file will cause the glyph representation for glyphid G_pool to use Unicode codepoint U+224B and the color represented by R-G-B value 0-0-160:

```
OPTIONS=glyph:G_pool/U+224B/0-0-160
```

The list of acceptable glyphid's can be produced by

```
nethack --glyphids
```

Individual NetHack glyphs can be specified using the G_prefix, or you can use an S_symbol for a glyphid and store the custom representation for all NetHack glyphs that would map to that particular symbol.

You will need to select a symset with a UTF8 handler to enable the display of the customizations, such as the Enhanced symset.

Configuring *NetHack* for Play by the Blind

NetHack can be set up to use only standard ASCII characters for making maps of the dungeons. This makes even the MS-DOS versions of *NetHack* (which use special line-drawing characters by default) completely accessible to the blind who use speech and/or Braille access technologies. Players will require a good working knowledge of their screen-reader's review features, and will have to know how to navigate horizontally and vertically character by character. They will also find the search capabilities of their screen-readers to be quite valuable. Be certain to examine this Guidebook before playing so you have an idea what the screen layout is like. You'll also need to be able to locate the PC cursor. It is always where your character is located. Merely searching for an @-sign will not always find your character since there are other humanoids represented by the same sign. Your screen-reader should also have a function which gives you the row and column of your review cursor and the PC cursor. These co-ordinates are often useful in giving players a better sense of the overall location of items on the screen. *NetHack* can also be compiled with support for sending the game messages to an external program, such as a text-to-speech synthesizer. If the "`#version`" extended command shows "external program as a message handler", your *NetHack* has been compiled with the capability. When compiling *NetHack* from source on Linux and other POSIX systems, define `MSGHANDLER` to enable it. To use the capability, set the environment variable `NETHACK_MSGHANDLER` to an executable, which will be executed with the game message as the program's only parameter.

The most crucial settings to make the game more accessible are:

symset:plain Load a symbol set appropriate for use by blind players.

menustyle:traditional This will assist in the interface to speech synthesizers.

nomenu_overlay Show menus on a cleared screen and aligned to the left edge.

number_pad A lot of speech access programs use the number-pad to review the screen. If this is the case, disable the number_pad option and use the traditional Rogue-like commands.

paranoid_confirmation:swim Prevent walking into water or lava.

accessiblemsg Adds direction or location information to messages.

spot_monsters Shows a message when hero notices a monster; combine with accessiblemsg.

mon_movement Shows a message when hero notices a monster movement; combine with spot_monsters and accessiblemsg.

autodescribe Automatically describe the terrain under the cursor when targeting.

mention_map Give feedback messages when interesting map locations change.

mention_walls Give feedback messages when walking towards a wall or when travel command was interrupted.

whatis_coord:compass When targeting with cursor, describe the cursor position with coordinates relative to your character.

whatis_filter:area When targeting with cursor, filter possible locations so only those in the same area (eg. same room, or same corridor) are considered.

whatis_moveskip When targeting with cursor and using fast-move, skip the same glyphs instead of moving 8 units at a time.

nostatus_updates Prevent updates to the status lines at the bottom of the screen, if your screen-reader reads those lines. The same information can be seen via the `#attributes` command.

showdamage Give a message of damage taken and how many hit points are left.

Global Configuration for System Administrators

If *NetHack* is compiled with the SYSCF option, a system administrator should set up a global configuration; this is a file in the same format as the traditional per-user configuration file (see above).

This file should be named sysconf and placed in the same directory as the other *NetHack* support files. The options recognized in this file are listed below. Any option not set uses a compiled-in default (which may not be appropriate for your system).

WIZARDS A space-separated list of user name who are allowed to play in debug mode (commonly referred to as wizard mode). A value of a single asterisk (*) allows anyone to start a game in debug mode.

SHELLERS A list of users who are allowed to use the shell escape command (‘`!`’). The syntax is the same as WIZARDS.

EXPLORERS A list of users who are allowed to use the explore mode. The syntax is the same as WIZARDS.

MSGHANDLER A path and filename of executable. Whenever a message-window message is shown, *NetHack* runs this program. The program will get the message as the only parameter.

MAXPLAYERS Limit the maximum number of games that can be running at the same time.

SUPPORT A string explaining how to get local support (no default value).

RECOVER A string explaining how to recover a game on this system (no default value).

SEDUCE 0 or 1 to disable or enable, respectively, the SEDUCE option. When disabled, incubi and succubi behave like nymphs.

CHECK_PLNAME Setting this to 1 will make the EXPLORERS, WIZARDS, and SHELLERS check for the player name instead of the user’s login name.

CHECK_SAVE_UID 0 or 1 to disable or enable, respectively, the UID (used identification number) checking for save files (to verify that the user who is restoring is the same one who saved).

The following four options affect the score file:

PERSMAX Maximum number of entries for one person.

ENTRYMAX Maximum number of entries in the score file.

POINTSMIN Minimum number of points to get an entry in the score file.

PERS_IS_UID 0 or 1 to use user names or numeric userids, respectively, to identify unique people for the score file.

HIDEUSAGE 0 or 1 to control whether the help menu entry for command line usage is shown or suppressed.

MAX_STATUENAME_RANK Maximum number of score file entries to use for random statue names (default is 10).

ACCESSIBILITY 0 or 1 to disable or enable, respectively, the ability for players to set S_pet_override and S_hero_override symbols in their configuration file.

PORTABLE_DEVICE_PATHS 0 or 1 Windows OS only, the game will look for all of its external files, and write to all of its output files in one place rather than at the standard locations.

DUMPLOGFILE A filename where the end-of-game dumplog is saved. Not defining this will prevent dumplog from being created. Only available if your game is compiled with DUMPLOG. Allows the following placeholders: `%%` — literal ‘`%`’
`%v` — version (eg. “`5.0.0-0`”)
`%u` — game UID

`%t` — game start time, UNIX timestamp format
`%T` — current time, UNIX timestamp format
`%d` — game start time, YYYYMMDDhhmmss format
`%D` — current time, YYYYMMDDhhmmss format
`%n` — player name
`%N` — first character of player name

LIVELOG A bit-mask of types of events that should be written to the *livelog* file if one is present. The sample *sysconf* file accompanying the program contains a comment which lists the meaning of the various bits used. Intended for server systems supporting simultaneous play by multiple players (to be clear, each one running a separate single player game), for displaying their game progress to observers. Only relevant if the program was built with LIVELOG enabled. When available, it should be left commented out on single player installations because over time the file could grow to be extremely large unless it is actively maintained.

CRASHREPORTURL If set to `https://www.nethack.org/links/cr-37BETA.html` and support is compiled in, brings up a browser window populated with the information needed to report a problem if the game panics or ends up in an internally inconsistent state, or if the #bugreport command is invoked.

10 Scoring

NetHack maintains a list of the top scores or scorers on your machine, depending on how it is set up. In the latter case, each account on the machine can post only one non-winning score on this list. If you score higher than someone else on this list, or better your previous score, you will be inserted in the proper place under your current name. How many scores are kept can also be set up when *NetHack* is compiled.

Your score is chiefly based upon how much experience you gained, how much loot you accumulated, how deep you explored, and how the game ended. If you quit the game, you escape with all of your gold intact. If, however, you get killed in the Mazes of Menace, the guild will only hear about 90 % of your gold when your corpse is discovered (adventurers have been known to collect finder's fees). So, consider whether you want to take one last hit at that monster and possibly live, or quit and stop with whatever you have. If you quit, you keep all your gold, but if you swing and live, you might find more.

If you just want to see what the current top players/games list is, you can type

```
nethack -s all
```

on most versions.

11 Explore mode

NetHack is an intricate and difficult game. Novices might falter in fear, aware of their ignorance of the means to survive. Well, fear not. Your dungeon comes equipped with an "explore" or "discovery" mode that enables you to keep old save files and cheat death, at the paltry cost of not getting on the high score list.

There are two ways of enabling explore mode. One is to start the game with the `-X` command-line switch or with the *playmode:explore* option. The other is to issue the '`#exploremode`' extended command while already playing the game. Starting a new game in explore mode provides your character with a wand of wishing in initial inventory; switching during play does not. The other benefits of explore mode are left for the trepid reader to discover.

Debug mode

Debug mode, also known as wizard mode, is undocumented aside from this brief description and the various "debug mode only" commands listed among the command descriptions. It is intended for tracking down problems within the program rather than to provide god-like powers to your character, and players who attempt debugging are expected to figure out how to use it themselves. It is initiated by starting the game with the `-D` command-line switch or with the *playmode:debug* option.

For some systems, the player must be logged in under a particular user name to be allowed to use debug mode; for others, the hero must be given a particular character name (but may be any role; there's no connection

between "wizard mode" and the *Wizard* role). Attempting to start a game in debug mode when not allowed or not available will result in falling back to explore mode instead.

12 Credits

The original *hack* game was modeled on the Berkeley UNIX *rogue* game. Large portions of this document were shamelessly cribbed from *A Guide to the Dungeons of Doom*, by Michael C. Toy and Kenneth C. R. C. Arnold. Small portions were adapted from *Further Exploration of the Dungeons of Doom*, by Ken Arromdee.

NetHack is the product of literally scores of people's work. Main events in the course of the game development are described below:

Jay Fenlason wrote the original *Hack*, with help from *Kenny Woodland*, *Mike Thome*, and *Jon Payne*.

Andries Brouwer did a major re-write while at Stichting Mathematisch Centrum (now Centrum Wiskunde & Informatica), transforming Hack into a very different game. He published the Hack source code for use on UNIX systems by posting that to Usenet newsgroup *net.sources* (later renamed *comp.sources*) releasing version 1.0 in December of 1984, then versions 1.0.1, 1.0.2, and finally 1.0.3 in July of 1985. Usenet newsgroup *net.games.hack* (later renamed *rec.games.hack*, eventually replaced by *rec.games.roguelike.nethack*) was created for discussing it.

Don G. Kneller ported *Hack* 1.0.3 to Microsoft C and MS-DOS, producing *PC Hack* 1.01e, added support for DEC Rainbow graphics in version 1.03g, and went on to produce at least four more versions (3.0, 3.2, 3.51, and 3.6; note that these are old *Hack* version numbers, not contemporary *NetHack* ones).

R. Black ported *PC Hack* 3.51 to Lattice C and the Atari 520/1040ST, producing *ST Hack* 1.03.

Mike Stephenson merged these various versions back together, incorporating many of the added features, and produced *NetHack* version 1.4 in 1987. He then coordinated a cast of thousands in enhancing and debugging *NetHack* 1.4 and released *NetHack* versions 2.2 and 2.3. Like Hack, they were released by posting their source code to Usenet where they remained available in various archives accessible via *ftp* and *uucp* after expiring from the newsgroup.

Later, Mike coordinated a major re-write of the game, heading a team which included *Ken Arromdee*, *Jean-Christophe Collet*, *Steve Creps*, *Eric Hendrickson*, *Izchak Miller*, *Eric S. Raymond*, *John Rupley*, *Mike Threepoint*, and *Janet Walz*, to produce *NetHack* 3.0c.

NetHack 3.0 was ported to the Atari by *Eric R. Smith*, to OS/2 by *Timo Hakulinen*, and to VMS by *David Gentzel*. The three of them and *Kevin Darcy* later joined the main *NetHack Development Team* to produce subsequent revisions of 3.0.

Olaf Seibert ported *NetHack* 2.3 and 3.0 to the Amiga. *Norm Meluch*, *Stephen Spackman* and *Pierre Martineau* designed overlay code for *PC NetHack* 3.0. *Johnny Lee* ported *NetHack* 3.0 to the Macintosh. Along with various other Dungeoneers, they continued to enhance the PC, Macintosh, and Amiga ports through the later revisions of 3.0.

Version 3.0 went through ten relatively rapidly released "patch-level" revisions. Versions at the time were known as 3.0 for the base release and variously as "3.0a" through "3.0j", "3.0 patchlevel 1" through "3.0 patchlevel 10", or "3.0pl1" through "3.0pl10" rather than 3.0.0 and 3.0.1 through 3.0.10; the three component numbering scheme began to be used with 3.1.0.

Headed by *Mike Stephenson* and coordinated by *Izchak Miller* and *Janet Walz*, the *NetHack Development Team* which now included *Ken Arromdee*, *David Cohrs*, *Jean-Christophe Collet*, *Kevin Darcy*, *Matt Day*, *Timo Hakulinen*, *Steve Linhart*, *Dean Luick*, *Pat Rankin*, *Eric Raymond*, and *Eric Smith* undertook a radical revision of 3.0. They re-structured the game's design, and re-wrote major parts of the code. They added multiple dungeons, a new display, special individual character quests, a new endgame and many other new features, and produced *NetHack* 3.1. Version 3.1.0 was released in January of 1993.

Ken Lorber, *Gregg Wonderly* and *Greg Olson*, with help from *Richard Addison*, *Mike Passaretti*, and *Olaf Seibert*, developed *NetHack* 3.1 for the Amiga.

Norm Meluch and *Kevin Smolkowski*, with help from *Carl Schelin*, *Stephen Spackman*, *Steve VanDevender*, and *Paul Winner*, ported *NetHack* 3.1 to the PC.

Jon W{tte and *Hao-yang Wang*, with help from *Ross Brown*, *Mike Engber*, *David Hairston*, *Michael Hamel*, *Jonathan Handler*, *Johnny Lee*, *Tim Lennan*, *Rob Menke*, and *Andy Swanson*, developed *NetHack* 3.1 for the Macintosh, porting it for MPW. Building on their development, *Bart House* added a Think C port.

Timo Hakulinen ported *NetHack* 3.1 to OS/2. *Eric Smith* ported *NetHack* 3.1 to the Atari. *Pat Rankin*, with help from *Joshua Delahunty*, was responsible for the VMS version of *NetHack* 3.1. *Michael Allison* ported *NetHack* 3.1 to Windows NT.

Dean Luick, with help from *David Cohrs*, developed *NetHack* 3.1 for X11. It drew the map as text rather than graphically but included `nh10.bdf`, an optionally used custom X11 font which has tiny images in place of letters and punctuation, a precursor of tiles. Those images don't extend to individual monster and object types, just replacements for monster and object classes (so one custom image for all "`a`" insects and another for all "`[`" armor and so forth, not separate images for beetles and ants or for cloaks and boots).

Warwick Allison wrote a graphically displayed version of *NetHack* for the Atari where the tiny pictures were described as "icons" and were distinct for specific types of monsters and objects rather than just their classes. He contributed them to the *NetHack Development Team* which rechristened them "tiles", original usage which has subsequently been picked up by various other games. *NetHack's* tiles support was then implemented on other platforms (initially MS-DOS but eventually Windows, Qt, and X11 too).

The 3.2 *NetHack Development Team*, comprised of *Michael Allison*, *Ken Arromdee*, *David Cohrs*, *Jessie Collet*, *Steve Creps*, *Kevin Darcy*, *Timo Hakulinen*, *Steve Linhart*, *Dean Luick*, *Pat Rankin*, *Eric Smith*, *Mike Stephenson*, *Janet Walz*, and *Paul Winner*, released version 3.2.0 in April of 1996.

Version 3.2 marked the tenth anniversary of the formation of the development team. In a testament to their dedication to the game, all thirteen members of the original *NetHack Development Team* remained on the team at the start of work on that release. During the interval between the release of 3.1.3 and 3.2.0, one of the founding members of the *NetHack Development Team*, *Dr. Izchak Miller*, was diagnosed with cancer and passed away. That release of the game was dedicated to him by the development and porting teams.

Version 3.2 proved to be more stable than previous versions. Many bugs were fixed, abuses eliminated, and game features tuned for better game play.

During the lifespan of *NetHack* 3.1 and 3.2, several enthusiasts of the game added their own modifications to the game and made these "variants" publicly available:

Tom Proudfoot and *Yuval Oren* created *NetHack++*, which was quickly renamed *NetHack−−* when some people incorrectly assumed that it was a conversion of the *C* source code to *C++*. Working independently, *Stephen White* wrote *NetHack Plus*. *Tom Proudfoot* later merged *NetHack Plus* and his own *NetHack−−* to produce *SLASH*. *Larry Stewart-Zerba* and *Warwick Allison* improved the spell casting system with the Wizard Patch. *Warwick Allison* also ported *NetHack* to use the Qt interface.

Warren Cheung combined *SLASH* with the Wizard Patch to produce *Slash'EM*, and with the help of *Kevin Hugo*, added more features. Kevin later joined the *NetHack Development Team* and incorporated the best of these ideas into *NetHack* 3.3.

The final update to 3.2 was the bug fix release 3.2.3, which was released simultaneously with 3.3.0 in December 1999 just in time for the Year 2000. Because of the newer version, 3.2.3 was released as a source code patch only, without any ready-to-play distribution for systems that usually had such.

(To anyone considering resurrecting an old version: all versions before 3.2.3 had a *Y2K* bug. The high scores file and the log file contained dates which were formatted using a two-digit year, and 1999's year 99 was followed by 2000's year 100. That got written out successfully but it unintentionally introduced an extra column in the file layout which prevented score entries from being read back in correctly, interfering with insertion of new high scores and with retrieval of old character names to use for random ghost and statue names in the current game.)

The 3.3 *NetHack Development Team*, consisting of *Michael Allison*, *Ken Arromdee*, *David Cohrs*, *Jessie Collet*, *Steve Creps*, *Kevin Darcy*, *Timo Hakulinen*, *Kevin Hugo*, *Steve Linhart*, *Ken Lorber*, *Dean Luick*, *Pat Rankin*, *Eric Smith*, *Mike Stephenson*, *Janet Walz*, and *Paul Winner*, released 3.3.0 in December 1999 and 3.3.1 in August of 2000.

Version 3.3 offered many firsts. It was the first version to separate race and profession. The Elf class was removed in preference to an elf race, and the races of dwarves, gnomes, and orcs made their first appearance in the game alongside the familiar human race. Monk and Ranger roles joined Archeologists, Barbarians, Cavemen,

Healers, Knights, Priests, Rogues, Samurai, Tourists, Valkyries and of course, Wizards. It was also the first version to allow you to ride a steed, and was the first version to have a publicly available web-site listing all the bugs that had been discovered. Despite that constantly growing bug list, 3.3 proved stable enough to last for more than a year and a half.

The 3.4 *NetHack Development Team* initially consisted of *Michael Allison*, *Ken Arromdee*, *David Cohrs*, *Jessie Collet*, *Kevin Hugo*, *Ken Lorber*, *Dean Luick*, *Pat Rankin*, *Mike Stephenson*, *Janet Walz*, and *Paul Winner*, with *Warwick Allison* joining just before the release of *NetHack* 3.4.0 in March 2002.

As with version 3.3, various people contributed to the game as a whole as well as supporting ports on the different platforms that *NetHack* runs on:

Pat Rankin maintained 3.4 for VMS.

Michael Allison maintained *NetHack* 3.4 for the MS-DOS platform. *Paul Winner* and *Yitzhak Sapir* provided encouragement.

Dean Luick, *Mark Modrall*, and *Kevin Hugo* maintained and enhanced the Macintosh port of 3.4.

Michael Allison, *David Cohrs*, *Alex Kompel*, *Dion Nicolaas*, and *Yitzhak Sapir* maintained and enhanced 3.4 for the Microsoft Windows platform. *Alex Kompel* contributed a new graphical interface for the Windows port. *Alex Kompel* also contributed a Windows CE port for 3.4.1.

Ron Van Iwaarden was the sole maintainer of *NetHack* for OS/2 the past several releases. Unfortunately Ron's last OS/2 machine stopped working in early 2006. A great many thanks to Ron for keeping *NetHack* alive on OS/2 all these years.

Janne Salmijärvi and *Teemu Suikki* maintained and enhanced the Amiga port of 3.4 after *Janne Salmijärvi* resurrected it for 3.3.1.

Christian "Marvin" Bressler maintained 3.4 for the Atari after he resurrected it for 3.3.1.

The release of *NetHack* 3.4.3 in December 2003 marked the beginning of a long release hiatus. 3.4.3 proved to be a remarkably stable version that provided continued enjoyment by the community for more than a decade. The *NetHack Development Team* slowly and quietly continued to work on the game behind the scenes during the tenure of 3.4.3. It was during that same period that several new variants emerged within the *NetHack* community. Notably sporkhack by Derek S. Ray, *unnethack* by Patric Mueller, *nitrohack* and its successors originally by Daniel Thaler and then by Alex Smith, and *Dynahack* by Tung Nguyen. Some of those variants continue to be developed, maintained, and enjoyed by the community to this day.

In September 2014, an interim snapshot of the code under development was released publicly by other parties. Since that code was a work-in-progress and had not gone through the process of debugging it as a suitable release, it was decided that the version numbers present on that code snapshot would be retired and never used in an official *NetHack* release. An announcement was posted on the *NetHack Development Team*'s official *nethack.org* website to that effect, stating that there would never be a 3.4.4, 3.5, or 3.5.0 official release version.

In January 2015, preparation began for the release of *NetHack* 3.6.

At the beginning of development for what would eventually get released as 3.6.0, the *NetHack Development Team* consisted of *Warwick Allison*, *Michael Allison*, *Ken Arromdee*, *David Cohrs*, *Jessie Collet*, *Ken Lorber*, *Dean Luick*, *Pat Rankin*, *Mike Stephenson*, *Janet Walz*, and *Paul Winner*. In early 2015, ahead of the release of 3.6.0, new members *Sean Hunt*, *Pasi Kallinen*, and *Derek S. Ray* joined the *NetHack Development Team*.

Near the end of the development of 3.6.0, one of the significant inspirations for many of the humorous and fun features found in the game, author Terry Pratchett, passed away. *NetHack* 3.6.0 introduced a tribute to him.

3.6.0 was released in December 2015, and merged work done by the development team since the release of 3.4.3 with some of the beloved community patches. Many bugs were fixed and some code was restructured.

The *NetHack Development Team*, as well as *Steve VanDevender* and *Kevin Smolkowski*, ensured that *NetHack* 3.6 continued to operate on various UNIX flavors and maintained the X11 interface.

Ken Lorber, *Haoyang Wang*, *Pat Rankin*, and *Dean Luick* maintained the port of *NetHack* 3.6 for MacOS.

Michael Allison, *David Cohrs*, *Bart House*, *Pasi Kallinen*, *Alex Kompel*, *Dion Nicolaas*, *Derek S. Ray* and *Yitzhak Sapir* maintained the port of *NetHack* 3.6 for Microsoft Windows.

Pat Rankin attempted to keep the VMS port running for *NetHack* 3.6, hindered by limited access. *Kevin Smolkowski* has updated and tested it for the most recent version of OpenVMS (V8.4 as of this writing) on Alpha and Integrity (aka Itanium aka IA64) but not VAX.

Ray Chason resurrected the MS-DOS port for 3.6 and contributed the necessary updates to the community at large.

In late April 2018, several hundred bug fixes for 3.6.0 and some new features were assembled and released as *NetHack* 3.6.1. The *NetHack Development Team* at the time of release of 3.6.1 consisted of *Warwick Allison*, *Michael Allison*, *Ken Arromdee*, *David Cohrs*, *Jessie Collet*, *Pasi Kallinen*, *Ken Lorber*, *Dean Luick*, *Patric Mueller*, *Pat Rankin*, *Derek S. Ray*, *Alex Smith*, *Mike Stephenson*, *Janet Walz*, and *Paul Winner*.

In early May 2019, another 320 bug fixes along with some enhancements and the adopted curses window port, were released as 3.6.2.

Bart House, who had contributed to the game as a porting team participant for decades, joined the *NetHack Development Team* in late May 2019.

NetHack 3.6.3 was released on December 5, 2019 containing over 190 bug fixes to *NetHack* 3.6.2.

NetHack 3.6.4 was released on December 18, 2019 containing a security fix and a few bug fixes.

NetHack 3.6.5 was released on January 27, 2020 containing some security fixes and a small number of bug fixes.

NetHack 3.6.6 was released on March 8, 2020 containing a security fix and some bug fixes.

NetHack 3.6.7 was released on February 16, 2023 containing a security fix and some bug fixes.

Development work for the major release to follow *NetHack* 3.6 began in 2015 around the same time as *NetHack* 3.6.0 was being released. That development work continued in parallel to each of the *NetHack* 3.6 releases from 2015 through 2023, and continued until the end of April 2026. For the first time, that development was shared publicly on GitHub and SourceForge as it occurred. It was done under the label NetHack-3.7 work-in-progress (WIP), although the version number for the next release had not yet been solidified.

Exposure of the development to the public brought many good things, and some challenges. People were able to observe and criticize changes and new features almost immediately, and they often did. The GitHub pull request system made it straightforward for people to contribute directly to development. Contributions resolved many, many bugs in the game and we thank all the contributors.

In early 2026, with the game development getting stable enough to consider initiating an official release, the devteam reviewed the nature and number of changes in the game. It was clear that there was sufficient depth and breadth to warrant a major release and version 5.0 was decided on. That's a new major release over 3.x, without opening up any ambiguity or confusion with existing variants that there might have been had it been released as version 4.0.

NetHack 5.0.0 was released on May 2, 2026.

The source code for *NetHack* 5.0.0 was modified and modernized to be compliant with the C99 standard. The 5.0.0 release contained over 3100 fixes, changes, and updated features.

NetHack 5.0 was the first version to replace the lex and yacc level and dungeon compilers of past versions, with a new Lua interpreter-based approach to provide those elements. Lua is also used for the quest texts in *NetHack* 5.0.0. The entire development team acknowledges the work done by *Pasi Kallinen* to make that happen. *Ray Chason*

At the time of the *NetHack* 5.0 release, the core *NetHack Development Team*, active and erstwhile, included *Warwick Allison*, *Michael Allison*, *Ken Arromdee*, *David Cohrs*, *Jessie Collet*, *Bart House*, *Kevin Hugo*, *Pasi Kallinen*, *Ken Lorber*, *Dean Luick*, *Pat Rankin*, *Derek S. Ray*, *Alex Smith*, *Patric Mueller*, *Mike Stephenson*, *Janet Walz*, *Paul Winner*.

Ken Lorber, *Pat Rankin*, *Patrick Mueller* and *Michael Allison* helped ensure that *NetHack* 5.0 would run on macOS.

Ingo Paschke somehow managed to revive a *NetHack* 5.0 port for the Amiga, using a cross-compiler on a modern platform to do so. His work was shared so others can straightforwardly produce *NetHack* 5.0 for the Amiga thanks to his efforts.

Ray Chason contributed the majority of maintenance work for the *NetHack* 5.0 MS-DOS port, including porting the curses interface to it. *Michael Allison* ensured that *NetHack* 5.0 core changes continued to work with the msdos port and keep it alive. Cross-compiling the MS-DOS port has helped make that possible and mostly painless.

People that contributed to the Windows port of *NetHack* 5.0 since the development of *NetHack* 5.0 began over eleven years ago, included *Michael Allison*, *David Cohrs*, *Bart House*, *Pasi Kallinen*, *Alex Kompel*, *Dion Nicolaas*, *Derek S. Ray* and *Yitzhak Sapir*.

With sadness, the devteam would like to acknowledge and remember the past contributions from the late *Ron Van Iwaarden*, who was the sole maintainer of *NetHack* for OS/2 for several past *NetHack* releases. Ron will be missed.

The official *NetHack* web site is maintained by *Ken Lorber* at `https://www.nethack.org`.

Special Thanks

On behalf of the *NetHack* community, thank you very much once again to *M. Drew Streib* and *Pasi Kallinen* for providing a public *NetHack* server at nethack.alt.org. Thanks to *Keith Simpson* and *Andy Thomson* for hardfought.org. Thanks to all those unnamed dungeoneers who invest their time and effort into annual *NetHack* tournaments such as *Junethack*, *The November NetHack Tournament*, and in days past, *devnull.net* (gone for now, but not forgotten).

Dungeoneers

From time to time, some depraved individual out there in netland sends a particularly intriguing modification to help out with the game. The *NetHack Development Team* sometimes makes note of the names of the worst of these miscreants in this, the list of Dungeoneers:

Adam Aronow
Alex Kompel
Alex Smith
Andreas Dorn
Andy Church
Andy Swanson
Andy Thomson
Ari Huttunen
Bart House
Benson I. Margulies
Bill Dyer
Boudewijn Waijers
Bruce Cox
Bruce Holloway
Bruce Mewborne
Cameron Root
Carl Schelin
Chris Russo
David Cohrs
David Damerell
David Gentzel
David Hairston
Dean Luick
Del Lamb
Derek S. Ray
Deron Meranda
Dion Nicolaas
Dylan O'Donnell
Eric Backus
Eric Hendrickson
Eric R. Smith
Eric S. Raymond
Erik Andersen
Fredrik Ljungdahl
Frederick Roeber
G. Branden Robinson
Gil Neiger
Greg Laskin
Greg Olson
Gregg Wonderly
Hao-yang Wang
Helge Hafting
Ingo Paschke
Irina Rempt-Drijfhout
Izchak Miller
J. Ali Harlow
Janet Walz
Janne Salmijärvi
Jean-Christophe Collet
Jeff Bailey
Jochen Erwied
John Kallen
John Rupley
John S. Bien
Johnny Lee
Jon W{tte
Jonathan Handler
Joshua Delahunty
Karl Garrison
Keizo Yamamoto
Keith Simpson
Ken Arnold
Ken Arromdee
Ken Lorber
Ken Washikita
Kestrel Gregorich-Trevor
Kevin Darcy
Kevin Hugo
Kevin Sitze
Kevin Smolkowski
Kevin Sweet
Lars Huttar
Leon Arnott
M. Drew Streib
Malcolm Ryan
Mark Gooderum
Mark Modrall
Marvin Bressler
Matthew Day
Merlyn LeRoy
Michael Allison
Michael Feir
Michael Hamel
Michael Meyer
Michael Sokolov
Mike Engber
Mike Gallop
Mike Passaretti
Mike Stephenson
Mikko Juola
Nathan Eady
Norm Meluch
Olaf Seibert
Pasi Kallinen
Pat Rankin
Patric Mueller
Paul Winner
Pierre Martineau
Ralf Brown
Ray Chason
Richard Addison
Richard Beigel
Richard P. Hughey
Rob Menke
Robin Bandy
Robin Johnson
Roderick Schertler
Roland McGrath
Ron Van Iwaarden
Ronnen Miller
Ross Brown
Sascha Wostmann
Scott Bigham
Scott R. Turner
Sean Hunt
Stephen Spackman
Stefan Thielscher
Stephen White
Steve Creps
Steve Linhart
Steve VanDevender
Teemu Suikki
Tim Lennan
Timo Hakulinen
Tom Almy
Tom West
Warren Cheung
Warwick Allison
Yitzhak Sapir