UNIX Administration Course

Version 1.0

mapleson@gamers.org
Tel: (+44) (0)1772 893297
Fax: (+44) (0)1772 892913
WWW: http://www.futuretech.vuurwerk.nl/

Detailed Notes for Day 3 (Part 1)

UNIX Fundamentals: Installing an Operating System and/or Software.

Installation Rationale.

Installing an OS is a common task for an admin to perform, usually often because of the acquisition of a new system or the installation of a new disk.

Although any UNIX variant should be perfectly satisfactory once it has been installed, sometimes the admin or a user has a particular problem which requires, for example, a different system configuration (and thus perhaps a reinstall to take account of any major hardware changes), or a different OS version for compatibility testing, access to more up-to-date features, etc. Alternatively, a serious problem or accidental mistake might require a reinstallation, eg. corrupted file system, damaged disk, or an unfortunate use of the rm command (recall the example given in the notes for Day 2, concerning the dangers of the 'find' command); although system restoration via backups is an option, often a simple reinstall is more convenient and faster.

Whatever the reason, an admin must be familiar with the procedure for installing an OS on the platform for which she/he is responsible.

Installation Interface and Tools.

Most UNIX systems have two interfaces for software installation: a high-level mode where an admin can use some kind of GUI-based tool, and a low-level mode which employs the command line shell. The GUI tool normally uses the command line version for the actual installation operations. In the case of SGI's IRIX, the low-level program is called 'inst', while the GUI interface to inst is called 'swmgr' - the latter can be activated from the 'Toolchest' on the desktop, or entered as a command. Users can also run swmgr, but only in 'read-only' mode, ie. non-root users cannot use inst or swmgr to install or remove software.

For general software installation tasks (new/extra applications, updates, patches, etc.) the GUI tool can normally be used, but for installing an OS, virtually every UNIX platform will require the admin to not only use the low-level tool for the installation, but also carry out the installation in a 'lower' (restricted) access mode, ie. a mode where only the very basic system services are operating: no user-related processes are running, the end-user GUI interface is not active, no network services are running, etc. For SGI's IRIX, this mode is called 'miniroot'.

Major updates to the OS are also usually carried out in miniroot mode - this is because a fully operational system will have services running which could be altered by a major OS change, ie. it

would be risky to perform any such change in anything but the equivalent of miniroot.

It is common for this restricted miniroot mode to be selected during bootup, perhaps by pressing the ESC key when prompted. In the case of SGI systems, the motherboard PROM chip includes a hard-coded GUI interface mechanism called ARCS which displays a mouse-driven menu on bootup. This provides the admin with a user-friendly way of performing low-level system administration tasks, eg. installing the OS from miniroot, running hardware diagnostics, accessing a simple PROM shell called a Command Monitor for performing low-level actions such as changing PROM settings (eg. which SCSI ID to treat as the system disk), etc.

Systems without graphics boards, such as servers, provide the same menu but in text-only form, usually through a VT or other compatible text display terminal driven from the serial port. Note that SGI's VisualWorkstation machine (an NT system) also uses the ARCS GUI interface - a first for any NT system (ie. no DOS at all for low-level OS operations).

Not many UNIX vendors offer a GUI menu system like ARCS for low-level tasks - SGI is one of the few who do, probably because of a historical legacy of making machines for the visual arts and sciences. Though the ARCS system is perhaps unique, after one one has selected 'Software Installation' the procedure progresses to a stage where the interface does become the more familiar text-based use of inst (ie. the text information just happens to be presented within a GUI-style window).

Very early UNIX platforms were not so friendly when it came to offering an easy method for installing the OS, especially in the days of older storage media such as 5.25" disks, magnetic tapes, etc. However, some vendors did a good job, eg. the text-only interface for installing HP-UX on Hewlett Packard machines (eg. HP9000/730) is very user-friendly, allowing the admin to use the cursor arrow keys to select options, activate tasks, etc. During installation, constantly updated information shows how the installation is progressing: current file being installed, number of files installed so far, number of files remaining, amount of disk space used up so far, disk space remaining, percentage equivalents for all these, and even an estimate of how much longer the installation will take before completion (surprisingly, inst doesn't provide this last piece of information as it is running, though one can make good estimates or find out how long it's going to take from a 3rd-party information source).

The inst program gives progress output equivalent to most of the above by showing the current software subsystem being installed, which sub-unit of which subsystem, and what percentage of the overall operation has been done so far.

Perhaps because of the text-only interface which is at the heart of installing any UNIX variant, installing an OS can be a little daunting at first, but the actual procedure itself is very easy. Once an admin has installed an OS once, doing it again quickly becomes second nature. The main reason the task can seem initially confusing is that the printed installation guides are often *too* detailed, ie. the supplied documents have to assume that the person carrying out the installation may know nothing at all about what they're doing. Thankfully, UNIX vendors have recognised this fact and so nowadays any such printed material also contains a summary installation guide for experts and those who already know the general methods involved - this is especially useful when performing an OS update as opposed to an original OS installation.

OS Source Media.

Years ago, an OS would be stored on magnetic tape or 5.25" disks. Today, one can probably state with confidence that CDROMs are used by every vendor. For example, SGI's IRIX 6.2 comes on 2 CDROMs; IRIX 6.5 uses 4 CDROMs, but this is because 6.5 can be used with any machine from SGI's entire current product line, aswell as many older systems - thus, the basic CD set must contain the data for all relevant systems even though an actual installation will only use a small subset of the data from the CDs (typically less than one CD's worth).

In the future, it is likely that vendors will switch to DVDs due to higher capacities and faster transfer rates.

Though a normal OS installation uses some form of original OS media, UNIX actually allows one to install an OS (or any software) via some quite unique ways. For example, one could copy the data from the source media (I shall assume CDROM) to a fast UltraSCSI disk drive. Since disks offer faster transfer rates and access times, using a disk as a source media enables a faster installation, as well as removing the need for swapping CDROMs around during the installation process. This is essentially a time-saving feature but is also very convenient, eg. no need to carry around many CDROMs (remember that after an OS installation, an admin may have to install extra software, applications, etc. from other CDs).

A completely different option is to install the OS using a storage device which is attached to a remote machine across a network. This may sound strange, ie. the idea that a machine without an OS can access a device on a remote system and use that as an OS installation source. It's something which is difficult but not impossible with PCs (I'm not sure whether a Linux PC would support this method). A low-level communications protocol called bootp (Internet Bootstrap Protocol), supported by all traditional UNIX variants, is used to facilitate communication across the network. As long as the remote system has been configured to allow another system to access its local device as a source for remote OS installation, then the remote system will effectively act as an attatched storage medium.

However, most admins will rarely if ever have to install an OS this way for small networks, though it may be more convenient for larger networks. Note that IRIX systems are supplied by default with the bootp service disabled in the /etc/inetd.conf file (the contents of this file controls various network services). Full details on how to use the bootp service for remote OS installation are normally provided by the vendor in the form of an online book or reference page. In the case of IRIX, see the section entitled, "Booting across the Network" in Chapter 10 of the online book, "IRIX Admin: System Configuration and Operation".

Note: this discussion does not explain every single step of installing an OS on an SGI system, though the method will be demonstrated during the practical session if time permits. Instead, the focus here is on management issues which surround an OS installation, especially those techniques which can ease the installation task. Because of the SGI-related technical site I run, I have already created extremely detailed installation guides for IRIX 6.2 [1] and IRIX 6.5 [2] which also include tables of example installation times (these two documents are included for future reference). The installation times obtained were used to conduct a CPU and CDROM performance analysis [3]. Certain lessons were learned from this analysis which are also relevant to installing an OS - these are explained later.

Installing an OS on multiple systems.

Using a set of CDs to install an OS can take quite some time (15 to 30 minutes is a useful

approximation). If an admin has many machines to install, there are several techniques for cutting the amount of time required to install the OS on all the machines.

The most obvious method is for all machines to install via a remote network device, but this could actually be very slow, limited partly by network speed but also by the way in which multiple systems would all be trying to access the same device (eg. CDROM) at the same time. It would only really be effective for a situation where the network was very fast and the device - or devices, there could be more than one - was also fast.

An example would be the company MPC; as explained in previous lectures, their site configuration is extremely advanced. The network they employ is so fast that it can saturate the typical 100Mbit Ethernet port of a modern workstation like Octane. MPC's storage systems include many high-end RAID devices capable of delivering data at hundreds of MB/sec rates (this kind of bandwidth is needed for editing broadcast-quality video and assuring that animators can load complete scene databases without significant delay).

Thus, the admin at MPC can use some spare RAID storage to install an OS on a system across the network. When this is done, the limiting factor which determines how long the installation takes is the computer's main CPU(s) and/or its Ethernet port (100MBit), the end result of which is that an installation can take mere minutes. In reality, the MPC admin uses an even faster technique for installing an OS, which is discussed in a moment.

At the time of my visit, MPC was using a high-speed crossbar switching 288Mbit/sec network (ie. multiple communications links through the routers - each machine could be supplied with up to 36MB/sec). Today they use multiple gigabit links (HiPPI) and other supporting devices. But not everyone has the luxury of having such equipment.

Disk Cloning [1].

If an admin only has a single machine to deal with, the method used may not matter too much, but often the admin has to deal with many machines. A simple technique which saves a huge amount of time is called 'disk cloning'. This involves installing an OS onto a single system ('A') and then copying (ie. cloning) the contents of that system's disk onto other disks. The first installation might be carried out by any of the usual means (CDROM, DAT, network, etc.), after which any extra software is also installed; in the case of SGIs, this would mean the admin starting up the system into a normal state of operation, logging in as root and using swmgr to install extra items. At this point, the admin may wish to make certain custom changes as well, eg. installing shareware/freeware software, etc. This procedure could take more than an hour or two if there is a great deal of software to install.

Once the initial installation has finished, then begins the cloning process. On SGIs, this is typically done as follows (other UNIX systems will be very similar if not identical):

1. Place the system disk from another system B into system A, installed at, for example, SCSI ID 2 (B's system disk would be on SCSI ID 1 in the case of SGIs; SCSI ID 0 is used for the SCSI controller). Bootup the system.

2. Login as root. Use fx to initialise the B disk to be a new 'root' (ie. system) disk; create a file system on it; mount the disk on some partition on A's disk such as /disk2.

3. Copy the contents of disk A to disk B using a command such as tar. Details of how to do this with example tar commands are given in the reference guides [1] [2].

4. Every system disk contains special volume header information which is required in order to allow it to behave as a bootable device. tar cannot copy this information since it does not reside on the main data partition of the disk in the form of an ordinary file, so the next step is to copy the volume header data from A to B using a special command for that purpose. In the case of SGIs, the relevant program is called dvhtool (device volume header tool).

5. Shut down system A; remove the B disk; place the B disk back into system B, remembering to change its SCSI ID back to 1. If further cloning is required, insert another disk into system A on SCSI ID 2, and (if needed) a further disk into system B, also set to SCSI ID 2. Reboot both systems.

6. System A will reboot as normal. At bootup time, although system B already has a kernel file available (/unix) because all the files will be recognised as new (ie. changed) system B will also create a new kernel file (/unix.install) and then bootup normally ready for login. Reboot system B once more so that the new kernel file is made the current kernel file.

At this stage, what one has effectively created is a situation comprising *two* systems as described in Step 1, instead of only one such system which existed before the cloning process. Thus, one could now repeat the process again, creating four systems ready to use or clone again as desired. Then eight, sixteen, thirty two and so on. This is exactly the same way biological cells divide, ie. binary fission. Most people are familiar with the idea that repeatedly doubling the number of a thing can create a great many things in a short space of time, but the use of such a technique for installing an operating system on many machines means an admin can, for example, completely configure over one hundred machines in less than five hours! The only limiting factor, as the number of machines to deal with increases, is the amount of help available by others to aid in the swapping of disks, typing of commands, etc. In the case of the 18 Indys in Ve24, the last complete reinstall I did on my own took less than three hours.

Note: the above procedure assumes that each cloning step copies one disk onto just a single other disk - this is because I'm using the Indy as an example, ie. Indy only has internal space for one extra disk. But if a system has the available room, then many more disks could be installed on other SCSI IDs (3, 4, 5, etc.) resulting in each cloning step creating three, four, etc. disks from just one. This is only possible because one can run multiple tar copy commands at the same time. Of course, one could use external storage devices to connect extra disks. There's no reason why a system with two SCSI controllers (Indigo2, O2, Octane, etc.) couldn't use external units to clone the system disk to 13 other disks at the same time; for a small network, such an ability could allow the reinstallation of the entire system in a single step!

Using a Backup Image.

If a system has been backed up onto a medium such as DAT tape, one could in fact use that tape for installing a fresh OS onto a different disk, as opposed to the more usual use of the tape for data restoration purposes.

The procedure would be similar to some of the steps in disk cloning, ie. install a disk on SCSI ID 2, initialise, and use tar to extract the DAT straight to the disk. However, the volume header information would have to come from the original system since it would not be present on the tape,

and only one disk could be written to at a time from the tape. Backup media are usually slower than disks too.

Installing a New Version of an OS (Major Updates).

An admin will often have to install updates to various OS components as part of the normal routine of installing software patches, bug fixes, new features, security fixes, etc. as they arrive in CD form from the vendor concerned. These can almost always be installed using the GUI method (eg. swmgr) unless specifically stated otherwise for some reason. However, if an admin wishes to change a machine which already has an OS installed to a completely new version (whether a newer version or an older one), then other issues must be considered.

Although it is perfectly possible to upgrade a system to a newer OS, an existing system will often have so much software installed with a whole range of configuration files, a straight upgrade to a new OS revision may not work very well. It would be successful, but what usually happens is that the admin has to resolve installation conflicts before the procedure can begin, which is annoyingly time wasting. Further, some changes may even alter some fundamental aspect of the system, in which case an upgrade on top of the existing OS would involve extra changes which an admin would have to read up on first (eg. IRIX 6.2 uses a completely different file system to IRIX 5.3: XFS vs. EFS).

Even if an update over an existing OS is successful, one can never really be sure that older files which aren't needed anymore were correctly removed. To an admin, the system would 'feel' as if the older OS was somehow still there, rather like an old layer of paint hidden beneath a new gloss. This aspect of OS management is perhaps only psychological, but it can be important. For example, if problems occurerd later, an admin might waste time checking for issues concerning the older OS which aren't relevant anymore, even though the admin theoretically knows such checks aren't needed.

Thus, a much better approach is to perform a 'clean' installation when installing a new OS. A typical procedure would be as follows:

1. Read all the relevant notes supplied with the new OS release so that any issues relevant to how the system may be different with the new OS version are known beforehand, eg. if any system services operate in a different way, or other factors (eg. new type of file system, etc.)

2. Make a full system backup of the machine concerned.

3. Identify all the key files which make the system what it is, eg. /etc/sys_id, /etc/hosts, and other configuration files/directories such as /var/named, /var/flexlm/license.dat, etc. These could be placed onto a DAT, floptical, ZIP, or even another disk. Items such as shareware/freeware software are probably best installed anew (read any documents relevant to software such as this too).

4. Use the appropriate low-level method to reinitialise the system disk. For SGI IRIX systems, this means using the ARCS bootup menu to select the Command Monitor, boot off of the OS CDROM and use the fx program to reinitialise the disk as a root disk, use mkfs to create a new file system (the old OS image is now gone), then reboot to access the 'Install System Software' option from the ARCS menu.

5. Install the OS in the normal manner.

6. Use the files backed up in step 3 to change the system so that it adopts its usual identity and configuration, baring in mind any important features/caveats of the new OS release.

This is a safe and reliable way of ensuring a clean installation. Of course, the installation data could come from a different media or over a network from a remote system as described earlier.

Time-saving Tips.

When installing an OS or software from a CDROM, it's tempting to want to use the fastest possible CDROM available. However, much of the process of installing software, whether the task is an OS installation or not, involves operations which do not actually use the CDROM. For example, system checks need to be made before the installation can begin (eg. available disk space), hundreds of file structures need to be created on the disk, installation images need to be uncompressed in memory once they have been retrieved from the CDROM, installed files need to be checked as the installation progresses (checksums), and any post-installation tasks performed such as compiling any system software indices.

As a result, perhaps 50% of the total installation time may involve operations which do not access the CDROM. Thus, using a faster CDROM may not speedup the overall installation to any great degree. This effect is worsened if the CPU in the system is particularly old or slow, ie. a slow CPU may not be able to take full advantage of an old CDROM, never mind a new one.

In order for a faster CDROM to make any significant difference, the system's CPU must be able to take advantage of it, and a reasonably large proportion of an installation procedure must actually consist of accessing the CDROM.

For example, consider the case of installing IRIX 6.5 on two different Indys - one with a slow CPU, the other with a better CPU - comparing any benefit gained from using a 32X CDROM instead of a 2X CDROM [3]. Here is a table of installation times, in hours minutes and seconds, along with percentage speedups.

	2X CDROM	32X CDROM	%Speedup
100MHz R4600PC Indy:	1:18:36	1:12:11	8.2%
200MHz R4400SC Indy:	0:52:35	0:45:24	13.7%

(data for a 250MHz R4400SC Indigo2 shows the speedup would rise to 15.2% - a valid comparison since Indy and Indigo2 are almost identical in system design)

In other words, the better the main CPU, the better the speedup obtained by using a faster CDROM.

This leads on to the next very useful tip for installing software (OS or otherwise)...

Temporary Hardware Swaps.

The example above divided the columns in order to obtain the speedup for using a faster CDROM, but it should be obvious looking at the table that a far greater speedup can be obtained by using a

better CPU:

	Using 200MHz R4400SC CPU instead of 100MHz R4600PC. (Percentage Speedup)
2X CDROM with Indy:	33.1%
32X CDROM with Indy:	37.1%

In other words, no matter what CDROM is used, an admin can save approximately a third of the normal installation time just by temporarily swapping the best possible CPU into the target system! And of course, the saving is maximised by using the fastest CDROM available too, or other installation source such as a RAID containing the CDROM images.

For example, if an admin has to carry out a task which would normally be expected to take, say, three hours on the target system, then a simple component swap could save over an hour of installation time. From an admin's point of view, that means getting the job done quicker (more time for other tasks), and from a management point of view that means lower costs and better efficiency, ie. less wages money spent on the admin doing that particular task.

Some admins might have to install OS images as part of their job, eg. performance analysis or configuring systems to order. Thus, saving as much time as possible could result in significant daily productivity improvements.

The Effects of Memory Capacity.

During the installation of software or an OS, the system may consume large amounts of memory in order to, for example, uncompress installation images from the CDROM, process existing system files during a patch update, recompile system file indices, etc. If the target system does not have enough physical memory, then swap space (otherwise known as virtual memory) will have to be used. Since software installation is a disk and memory intensive task, this can massively slow down the installation or removal procedure (the latter can happen too because complex file processing may be required in order to restore system files to an earlier state prior to the installation of the software items being removed).

Thus, just as it can be helpful to temporarily swap a better CPU into the target system and use a faster CDROM if available, it is also a good idea to ensure the system has sufficient physical memory for the task.

For example, I once had cause to install a large patch upgrade to the various compiler subsystems on an Indy running IRIX 6.2 with 64MB RAM [1]. The update procedure seemed to be taking far too long (15 minutes and still not finished). Noticing the unusually large amount of disk activity compared to what I would normally expect, ie. noise coming from the disk, I became suspicious and wondered whether the installation process was running out of memory. A quick use of gmemusage showed the available memory to be very low (3MB) implying that memory swapping was probably occurring. I halted the update procedure (easy to do with IRIX) and cancelled the installation. After upgrading the system temporarily to 96MB RAM (using 32MB from another Indy) I ran the patch again. This time, the update was finished in less than one minute! Using gmemusage showed the patch procedure required at least 40MB RAM free in order to proceed without resorting to the use of swap space.

Summary.

1. Before making any major change to a system, make a complete backup just in case something goes wrong. Read any relevant documents supplied with the software to be installed, eg. release notes, caveats to installation, etc.

2. When installing an OS or other software, use the most efficient storage media available if possible, eg. the OS CDs copied onto a disk. NB: using a disk OS image for installation might mean repartitioning the disk so that the system regards the disk as a bootable device, just like a CDROM. By default, SCSI disks do not have the same partition layout as a typical CDROM. On SGIs using IRIX, the fx program is used to repartition disks.

3. If more than one system is involved, use methods such as disk cloning to improve the efficiency of the procedure.

4. If possible, temporarily swap better system components into the target system in order to reduce installation time and ensure adequate resources for the procedure (better CPU, lots of RAM, fastest possible CDROM).

Caution: item 4 above might not be possible if the particular set of files which get installed are determined by the presence of internal components. In the case of Indy, installing an R5000 series CPU would result in the installation of different low-level bootup CPU-initialisation libraries compared to R4600 or R4400 (these latter two CPUs can use the same libraries, but any R5000 CPU uses newer libraries). Files relevant to these kinds of issues are located in directories such as /var/sysgen.

Patch Files.

Installing software updates to parts of the OS or application software is a common task for admins. In general, patch files should not be installed unless they are needed, but sometimes an admin may not have any choice, eg. for security reasons, or Y2K compliance.

Typically, patch updates are supplied on CDs in two separate categories (these names apply to SGIs; other UNIX vendors probably use a similar methodology):

1. Required/Recommended patches.

2. Fix-on-Fail Patches.

Item 1 refers to patches which the vendor suggests the admin should definitely install. Typically, a CD containing such patches is accessed with inst/swmgr and an automatic installation carried out, ie. the admin lets the system work out which of the available required/recommended patches should be installed. This concept is known as installing a 'patch set'. When discussing system problems or issues with others (eg. technical support, or colleagues on the Net), the admin can then easily describe the OS state as being a particular revision modified by a particular dated patch set, eg. IRIX 6.5 with the April 1999 Patch Set.

Item 2 refers to patches which only concern specific problems or issues, typically a single patch file for each problem. An admin should not install such patches unless they are required, ie. they are selectively installed as and when is necessary. For example, an unmodified installation of IRIX 6.2

contains a bug in the 'jot' editor program which affects they way in which jot accesses files across an NFS-mounted directory (the bug can cause jot to erase the file). To fix the bug, one installs patch number 2051 which is shown in the inst/swmgr patch description list as 'Jot fix for mmapping', but there's no need to install the patch if a machine running 6.2 is not using NFS.

Patch Inheritance.

As time goes by, it is common for various bug fixes and updates from a number of patches to be brought together into a 'rollup' patch. Also, a patch file may contain the same fixes as an earlier patch plus some other additional fixes. Two issues arise from this:

1. If one is told to install a patch file of a particular number (eg. advice gained from someone on a newsgroup), it is usually the case that any later patch which has been declared to be a replacement for the earlier patch can be used instead. This isn't always the case, perhaps due to specific hardware issues of a particular system, but in general a fix for a problem will be described as 'install patch <whatever> or later'. The release notes for any patch file will describe what hardware platforms and OS revisions that patch is intended for, what patches it replaces, what bugs are fixed by the patch (official bug code numbers included), what other known bugs still exist, and what workarounds can be used to temporarily solve the remaining problems.

2. When a patch is installed, a copy of the effected files prior to installation, called a 'patch history', is created and safely stored away so that if ever the patch has to be removed at a later date, the system can restore the relevant files to the state they were in before the patch was first installed. Thus, installing patch files consumes disk space - how much depends on the patch concerned. The 'versions' command with the 'removehist' option can be used to remove the patch history for a particular patch, recovering disk space, eg.:

   ```
   versions removehist patchSG0001537
   ```

   would remove the patch history file for patch number 1537. To remove all patch histories, the command to use is:

   ```
   versions removehist "*"
   ```

Conflicts.

When installing patches, especially of the Fix-on-Fail variety, an admin can come across a situation where a patch to be installed (A) is incompatible with one already present on the system (B). This usually happens when an earlier problem was dealt with using a more up-to-date patch than was actually necessary. The solution is to either remove B, then install an earlier but perfectly acceptable patch C and finally install A, or find a more up-to-date patch D which supersedes A and is thus compatible with B.

Note: if the history file for a patch has been removed in order to save disk space, then it will not be possible to remove that patch from the system. Thus, if an admin encounters the situation described above, the only possible solution will be to find the more up-to-date patch D.

Exploiting Patch File Release Notes.

The release notes for patches can be used to identify which patches are compatible, as well ascertain other useful information, especially to check whether a particular patch is the right one an admin is looking for (patch titles can sometimes be somewhat obscure). Since the release notes exist on the system in text form (stored in /usr/relnotes), one can use the grep command to search the release notes for information by hand, using appropriate commands. The commands 'relnotes' and 'grelnotes' can be used to view release notes.

relnotes outputs only text. Without arguments, it shows a summary of all installed products for which release notes are available. One can then supply a product name - relnotes will respond with a list of chapter titles for that product. Finally, specifying a product name and a chapter number will output the actual text notes for the chosen chapter, or one can use '*' to display all chapters for a product. grelnotes gives the same information in a browsable format displayed in a window, ie. grelnotes is a GUI interface to relnotes. See the man pages for these commands for full details.

relnotes actually uses the man command to display information, ie. the release notes files are stored in the same compressed text format ('pack') used by online manual pages (man uses the 'unpack' command to decompress the text data). Thus, in order to grep-search through a release notes file, the file must first be uncompressed using the unpack command. This is a classic example of where the UNIX shell becomes very powerful, ie. one could write a shell script using a combination of find, ls, grep, unpack and perhaps other commands to allow one to search for specific items in release notes.

Although the InfoSearch tool supplied with IRIX 6.5 allows one to search release notes, IRIX 6.2 does not have InfoSearch, so an admin might decide that writing such a shell script would prove very useful. Incidentally, this is exactly the kind of useful script which ends up being made available on the Net for free so that anyone can use it. For all I know, such a script already exists. Over time, entire collections of useful scripts are gathered together and eventually released as freeware (eg. GNU shell script tools). An admin should examine any such tools to see if they could be useful - a problem which an admin has to deal with may already have been solved by someone else two decades earlier.

Patch Subsystem Components.

Like any other software product, a patch file is a software subsystem usually containing several sub-units, or components. When manually selecting a patch for installation, inst/swmgr may tag all sub-units for installation even if certain sub-units are not applicable (this can happen for an automatic selection too, perhaps because inst selects all of a patch's components by default). If this happens, any conflicts present will be displayed, preventing the admin from accidentally installing unwanted or irrelevant items. Remember that an installation cannot begin until all conflicts are resolved, though an admin can override this behaviour if desired.

Thus, when manually installing a patch file (or files), I always check the individual sub-units to see what they are. In this way, I can prevent conflicts from arising in the first place by not selecting subsystems which I know are not relevant, eg. 64bit libraries which aren't needed for a system with a 32bit memory address kernel like Indy (INFO: all SGIs released after the Indigo R3000 in 1991 do 64bit processing, but the main kernel file does not need to be compiled using 64bit addressing extensions unless the system is one which might have a very large amount of memory, eg. an Origin2000 with 16GB RAM). Even when no conflicts are present, I always check the selected components to ensure no 'older version' items have been selected.

References:

1. Disk and File System Administration:

   `http://www.futuretech.vuurwerk.nl/disksfiles.html`

2. How to Install IRIX 6.5:

   `http://www.futuretech.vuurwerk.nl/6.5inst.html`

3. SGI General Performance Comparisons:

   `http://www.futuretech.vuurwerk.nl/perfcomp.html`