A
Northern Illinois University
Academic Computing Services
Workshop

UNIX Basics for Superusers

Michael G. Prais
Swen Parson 120
753-1057

System Dynamics: Processes

Using UNIX requires interacting with a process called the *shell*
which interprets the command you type at a terminal.

The system itself starts when cycling the power causes
the processor to read and execute instructions
in a certain area of its memory.
The manufacturer permanently placed these instructions
on *read only memory (ROM)*.
After checking the hardware integrity,
the instructions typically require the workstation to ask
from which storage device should the next set of instructions be read.
The typical reply after the system has been initialized
is to read the *boot block* of the root partition of a disk drive.
The instructions on this section of the disk drive require the workstation
to load the file */boot* from the root partition of the disk drive
which can then load the UNIX kernel, */vmunix* or */unix* .

uptime	Displays the lifetime of the system and its load during the last minute, during the last five minutes, and during the last fifteen minutes.

The *kernel* provides all basic services for the system
that include memory access, device access, and processor access.
The kernel manages these resources
by allocating regions of memory not used by itself

to contain alternative processor instructions,
by requiring that any alternative instructions
request device access through the kernel,
by selecting which alternative instructions are to be executed next,
by interrupting the processor every 100 milliseconds to retain control.

vmstat 5 5	Displays the average process, memory, page, disk, interrupt, and processor activity, then the incremental values every 5 seconds for 5 times.

It is most useful to capture this information from time to time
when the system is quiet
to get a baseline from which to compare system load.

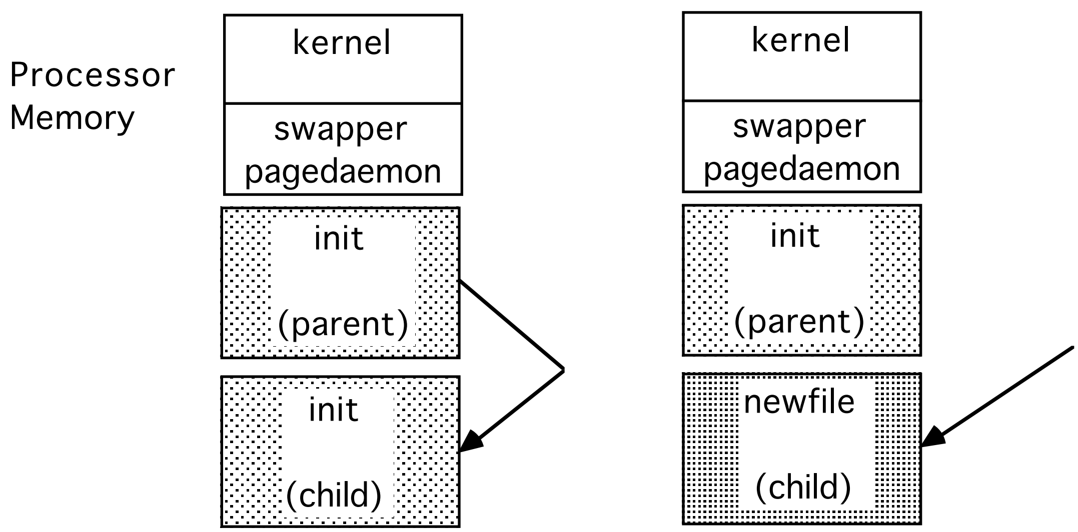
A region of memory managed by the kernel is called a *process*.
This assigned region of memory contains executable instructions,
assigned storage for variables,
a *stack* for transferring information between routines,
and administrative run-time information used by the kernel,
as well as information maintained by the kernel at all times.
The kernel tracks these processes with a *process table*.
Processes are identified with the following parameters.

Process ID
Parent process ID
Real user ID for accounting
Real group ID for accounting
Effective user ID for file access
Effective group ID for file access
Controlling terminal
Priority
Current status
Process memory map

Other tables track the files that a process has open.

pstat -T	Displays usage of file, inode, process, and swap tables used by kernel.
iostat 5 5	Displays character device transfer activity, kilobytes per second transferred for disks, milliseconds per seek, and the distribution of system time.

The kernel loads the executable file /etc/init into memory
as the first process.
The *init* process (and its progeny) generate each subsequent process
by requesting that the kernel make a copy of the original (parent) process
in another process space.



The kernel eventually allows the new (child) process to execute
the instructions that follow the request to duplicate,
and it requests the kernel to overwrite its process space
with some other executable file.

The kernel designates part of itself called the *swapper* as *process 0,*
init which is the first true process as *process 1* ,
and another part of itself called the *pagedaemon* as *process 2*.
UNIX is able to manage demands by processes for memory
that cumulatively exceed the physical memory on the system.

The pagedaemon serves to move
small, recently unused sections of a process
between memory and the UNIX *swap partition* on a disk drive
to provide temporary space for these processes to execute.
The swapper serves to move whole processes between memory
and the UNIX *swap partition* when the pagedaemon is excessively active.

pstat -s	Displays available swap space.

While in a single-user state, the *init* process initializes the system
by starting the command interpreter */bin/sh*
to run several configuration scripts.
Under System V the *init* process starts the processes
listed in */etc/inittab*.

The *runtime configuration* scripts such as
/etc/boot, */etc/rc.boot*, */etc/rc*, and */etc/rc.local*
(*/etc/brc*, */etc/bcheckrc*, and */etc/checklist* for System V)
allow the initialization to be customized without recompiling */etc/init*.

The script /etc/rc.boot sets the system name and
runs */etc/fsck -p* before multi-user initializations.

more /etc/rc.boot	Displays the contents of /etc/rc.boot.

The script */etc/rc* is run after single-user initializations
and before multiuser-user initializations.
This script mounts local file systems,
starts the command interpreter */bin/sh* to run the script */etc/rc.local*,
starts standard daemons,
preserves editor files,
clears */tmp* space,
starts system accounting,
and starts the network.

more /etc/rc	Displays the contents of */etc/rc*.

The output from most commands is redirected
to */dev/console* for viewing or to */dev/null* for disposal.

The script */etc/rc.local* mounts remote, network file systems, starts local daemons (servers), and does any local administration.

more /etc/rc.local	Displays the contents of /etc/rc.local.

Daemons are background system processes that provide and maintain necessary system resources and operations.

swapper	Moves whole processes between memory and storage.
pagedaemon	Moves small parts of processes between memory and storage.
update	Synchronizes the kernel and file systems.
syslogd	Manages and distributes system status and error messages.
cron	Starts *timered* processes.
lpd	Queues and dispatches print requests.
sendmail	Queues and dispatches mail requests.
comsat	Notifies users (found in */etc/utmp*) of the arrival of mail.
talkd	Handles talk requests.
inetd	Watches network ports and starts daemons listed in */etc/inetd.conf* or */etc/services*.
telnetd	Handles interactive requests for command execution from users on the network.
ftpd	Handles error-checked file transfer requests.
nfsd	Handles client requests for file access.
biod	Handles network file (block) input/output.
rwhod	Maintains a list of remote network users in /usr/spool/rwho/whod.*hostname*.
rexecd	Handles remote execution requests.
rlogind	Handles remote logins.
rshd	Handles remote shell requests for *rsh*, *rcmd*, and *rcp*.
timed	Maintains a network clock.
routed	Maintains a dynamic network routing table.
gated	Handles network gateway routing.
named	Handles domain name service.

The *ps* command displays process status for the system.

ps	Displays a list of process associated with your effective user ID and your controlling terminal.

PID	Process ID
PPID	Parent Process ID
TT	Controlling Terminal
CPU	User and System Processor Usage
STAT	Process Status R - runnable T - stopped P - Page wait D - Disk (short term) wait S - Sleeping less than 20 sec I - Idle more than 20 sec Z - Zombie Awaiting interment
<defunct>	exited but not waited by parent
<exiting>	blocked trying to exit
TIME	time on the system
COMMAND	Process Name

ps -g	Displays a list of all your processes with information about each.
ps -x	Displays only those processes without controlling terminals.
ps -aux	Displays a list of all processes with information about each user. USER %CPU %MEM SZ RSS START
ps -ut*tty*	Displays only those processes controlled by terminal *tty* (as per ps) with information about the user.

The *init* process puts the system into a multi-user state
by starting processes to service input/output terminals.
These terminals are called *ttys* ("tee-tee-whys")
ever since the origin of UNIX when all terminals were *teletypes*.
The information about which terminals to start and their characteristics
is kept in the file /etc/ttytab.

more /etc/ttytab Displays the list of terminals available with the program to initially run on each, the expected terminal type, whether the program should be run initially, whether the supervisor can use the terminal, and a comment preceded by an octothorpe (#).

The file /etc/ttys contains similar information which is generated automatically
by init and should not be edited.
Once /etc/ttytab is edited, init can be forced to reread it
by sending process 1 (init) the *hangup* signal.

#kill -HUP 1 Reads a new /etc/ttytab.

The most common program to run on a terminal is /etc/getty
although any process with input and output can be run.
These processes are listed in the output of process status (ps)
for those terminals that are not being used by users.

ps -a l grep getty Displays the list of getty processes.

The *getty* command waits for activity on the terminal line,
sets the line characteristics, sends out the *login:* prompt,
reads the user response, and executes the /etc/login command.
The line characteristics are stated in the /etc/gettytab file
or the /etc/gettydefs (System V) file.

more /etc/gettytab Displays the names and characterisitics of various line types.

The settings in /etc/gettytab should be a minimal set necessary for communications; more extensive settings can be associated with the terminal types found in the /etc/termcap file.

The *login:* prompt can be customized within /etc/termcap.
Sending a Break or NULL to /etc/getty resets the line characteristics
to those of the next (nx) line type pointed to by the current line type.
These files are described in the section on Communications.

The *login* command requests a password,
verifies the username and password combination in the /etc/passwd file,
updates accounting, add login entries in /etc/utmp and /usr/adm/wtmp,
prints the *message-of-the-day* from /etc/motd,
announces the existence of mail if there is any in /usr/spool/mail,
and changes the entry in /usr/adm/lastlog for that user.
The existence of the file /etc/nologin prevents access to the system.
When a getty finds this file, it displays its contents and exits.

more /etc/passwd	Displays user login information.

The /etc/passwd file is readable by all.
It is a security risk: when this file is writeable by anyone but its owner,
the superuser, when there are duplicate userids,
or when any fields are empty.
The system usernames *nobody*, *daemon*, *bin*, *kmem*, and *tty*
exist to own and organize system files;
they should have a password such as * that does not allow access.
The *nobody* username has a userid 65534 to provide a maximum userid.

more /etc/motd	Displays the message of the day.

The /etc/utmp file contains information about
the terminal, username, host, and time of users currently on the system.
The times are not readable text.

last -f /etc/utmp | more	Displays terminal, username, and host of users currently on the system with the offset into the file.

Information about terminal, username, host,
and times for user logins and logouts is placed in the /var/adm/wtmp file
when a user exits the system.
The times are not readable text.

last -f /var/adm/wtmp | more

Displays terminal, username, and host
for user logins and logouts
with the offset into the file.

The file /var/adm/wtmp should be cleared to avoid excessive size.

#cat /dev/null > /var/adm/wtmp Eliminates the contents of a file.

The /var/adm/lastlog file contains information about
terminal, host, and time of last login for each user ID.
The times of the last logins are not readable text.

od -s /var/adm/lastlog | more

Displays an array of terminal, host, and
time of last login for each user ID
with the offset into the file.

last *your_username* Displays the last login(s) for your username
from the /var/adm/wtmp file.

The *login* command also sets up the user login environment.
The environment consists of
the list of directories checked for executable files (PATH)
set to *:/usr/ucb:/bin:/usr/bin*,
the terminal type (TERM) from /etc/ttytab,
the username (USER) from /etc/passwd,
the user ID (UID) and group ID (GID) from /etc/passwd,
the working directory (HOME) from /etc/passwd,
and the command interpreter (SHELL) from /etc/passwd.
The variables set for the user login environment
are traditionally labeled with uppercase names.

setenv Displays your login environment.

The absence of anything before the first colon in PATH
indicates the current directory.
Delete the leading colon in the PATH for all users and
place a colon at the end of the PATH for users except the superuser.
The current directory preceding any system directories in a PATH or path
is a security risk a user expecting to run a system program
could instead run a similarly named program in some current directory
left there by an unscrupulous user.

The *login* command finally starts for the user
the command interpreter listed in the /etc/passwd file.
Any command can be used in place of the shell
to simply display or request information and then exit.
The shell sets its own environment variables.
The variables *term*, *path*, *user*, and *home* are set from the corresponding
login variables.
The shell maintains agreement between these shell and login variables
whenever either one is changed.

set	Displays the C shell environment variables.

Note the difference in the formats of the list of words in path and PATH.

The C shell also initializes several other variables.

prompt	Is displayed when the shell is waiting input.
argv[]	Is the list of arguements that follow a command.
cwd	Is the current working directory.
cdpath	Is the list of directories that cd can reach.
status	Is the completion status of the last command. where zero indicates success.
shell	Is the command interpreter used.

Whenever the C shell starts, it executes the commands in the hidden file .cshrc
in the user home directory.
The .cshrc file is the appropriate place to set shell variables and aliases.

more .cshrc	Displays your C shell runtime configuration.

Whenever the *C shell* is started by the *login* command,
it executes the .login file from the user home directory
after it executed the .cshrc file.
The .login file is the appropriate place to set up the terminal (tset)
to set (setenv) the login environment variables,
to set the default file creation permissions (umask),
and to run any initial or informative programs.

more .login	Displays your C shell login file.

When the user exits the login shell,
the C shell executes the commands in the .logout file
and adds an entry to /usr/adm/wtmp.

Initializing User Accounts

Setting up a user account is a simple sequence of steps.

more /etc/group	Displays the contents of /etc/group. *grpname*:*:*GID*: *username,username*
#cp /etc/group /etc/group-	Copies the original version of /etc/group.
#vi /etc/group	Edits the /etc/group file to identify or add a group for the new user.
grpck	Checks the integrity of /etc/group.
more /etc/passwd	Displays the contents of /etc/passwd.

The /etc/passwd file contains seven fields separated by colons.

username:passwd:UID:Default GID:name,office,phone,home:
home dir:login shell

The username can be upto eight characters.

The password is encrypted.

Placing any character(s) in the password field effectively stops access.
Regularly check that all usernames have passwords.

#passwd *username* Allows the superuser to set a user password.

Only the first eight characters of a password are checked.

The user IDs must be unique--especially zero for the superuser.
There are several system IDs such as *root*, *bin*, *sys*, *daemon*, and *uucp* with user ID values less than 10 or so.
Create a system in which IDs less than 20 are system IDs,
IDs less than 100 are used by application administrators,
and applications greater than 100 are user IDs.
It is often good security to create an ID of 65534 (-2) for *nobody* so that the largest value for the user ID cannot be created arbitrarily.

```
awk -F: 'BEGIN {maxuid = 0}
{if ( maxuid < $3 && $3 < 65534 ) {maxuid = $3}}
END {print maxuid}' /etc/passwd
```

Finds the largest user ID in /etc/passwd.

The default group ID in the /etc/passwd file is the first value set for the user on login; it should be the most likely group from which the user accesses files of other users.
The default group ID is used as the real group ID for accounting.
The group ID can be changed with the *newgroup* command, but it is often more convenient to add the user to the group within the /etc/groups file.

The information field in a /etc/passwd record should consist of four subfields separated by commas: full name, location, phone, and home.
This information is used by the *finger* command to identify users.

grep *username* /etc/passwd

Displays your password record.

finger *your_username* Displays information from your password record and .plan and .project files for each terminal used.

The shell field can be changed by the user with the *chsh* command to /bin/sh or /bin/csh.

chfn *your_name* Changes the text that describes you in the /etc/passwd file.

#vipw Safely edits the /etc/passwd file.

pwck Checks the /etc/passwd file integrity.

The /etc/passwd file is expected to be readable by all.

#mkdir /home/*username* Creates a home directory for the user.

Make copies of the default startup configuration files from /usr/lib
(Cshrc, Login, Profile, Logout, Exrc, and Mailrc),
place them in /usr/local/startup as .cshrc and so on,
and edit them for your specific system.
Check that the umask is set to 027 and
that the PATH or path does have the current directory
prior to system directories.

#cp /usr/local/startup/.[a-z]* /home/*username*

Copy the startup configuration files into the user's home directory.

Be sure to use the construction *.[a-z]** and not *.** or *.?** to pick up these files and not the whole directory (.) and parent directory (..) .

Since the superuser created these files,
their ownership and permissions must be changed
to allow the new user to access them.

#chown -R *username.grpname* /home/*username*

Changes the ownerships of the user's home directory from the of the superuser.

#chmod 755 /home/*username*

Changes the permissions of the user's home directory to allow only the user to create and delete files.

#chmod 644 /home/*username*/.[a-z]*	Allows only the user to change these files.

Check that the user's home directory exists.

cd /home/*username*	Change the current directory to the newly created directory.
pwd	Display the current directory.
ls -lagd	Display the ownership and permissions of the newly created directory.
ls -lag /home/*username*	Display the ownership and permissions of all the files.

Set up any aliases for the user's username in the mail system.

#cp /usr/lib/aliases /usr/lib/aliases-

Duplicates the mail alias file.

#vi /usr/lib/aliases	Safely edits the mail alias file in the format: *alias: username.*

more /usr/lib/aliases — Displays the mail alias file.
#newaliases — Reconstructs the system alias table.

Removing User Files

When a user is no longer active on your system,
you can remove the user's files,
but you should never remove the user ID record from /etc/passwd.
The user files always retain the user ID:
if you ever need to replace a file or files,
the user ID will never be in use and will always be available.
It is a good practice to replace the encrypted password of the user
with the current date to remind yourself that the account was closed.

grep *username* /etc/passwd — Displays the password record for username.

find / -user *username* -print > userfiles

Creates a list of all files owned by a user.

more userfiles — Lists the filename in *userfiles* and allows editing the list.

These files should include items*/home/username* and
/var/spool/mail/username.

All user files except the home directory with a .forward file
should be saved to tape and then removed from the file system.

tar cvf tarfile -I userfiles — Archives to tarfile all files included in userfiles.

tar tf tarfile — Displays the archived filenames.

#find / -user *username* -name \! ~*username*
-name \! ~*username/*.forward -print -exec rm {} \;

Removes all files owned by username

except the user's home directory and mail forwarding file.

ls ~*username/*.forward — Checks the existence of a .forward file.

echo *mail_address* > ~*username/*.forward

Creates directions to forward user mail.

chown *username.grpname* ~*username/*.forward

Gives the user ownership of the .forward file.

chmod a+r ~/.forward — Makes the .forward file readable by all.

Timed Activities

The *at* command can be used to run various administrative operations at specific times rather than at the present.

at -m 2:30 pm Fri echo "Output from at command" > at.out ls >> at.out CTRL D	Schedules a process for later and mails notification of completion.
at -m now + 20 minutes finger > at.out2 printenv >> at.out2 set >> at.out2 CTRL D	Schedules a process for later and mails notification of completion.
atq	Lists the *at* jobs.
atrm -i *your_username*	Prompts with a list of jobs that could be removed from the *at* queue.

The *at* command accepts either input from the keyboard or as a file.

Placing an *at* comand in the file with *next* as one of the arguments provides a mechanism to restart the command on a regular basis.

at -m Mon	Schedules another *at* for next Monday.
at -m next Mon	
echo “Hi Ho Hi Ho ...”	
CTRL D	

The *crontab* command is the more appropriate way to schedule a recurrent process.

crontab -e	Opens a file with the visual editor that can be used to schedule processes.

cc
Send mail on every quarter hour of 3 PM on Fridays.
0,15,30,45 15 * * 5 ls 2>&1 | mail *your_username*% *subject_message*
Esc
:x

The Bourne shell construct *2>&1* redirects error output to standard output.

The format of the crontab file is listed below.

minutes hours dates months weekdays commands%input

A wildcard (*) can be used and commas can be used in a list of values.
Any text following the percent (%) sign is taken as input to the command.
In this case mail prompts for a subject.

crontab -l	Lists your cron table.
crontab	Accepts keyboard input and overwrites your current crontab
CTRL D	unless you interrupt it (Ctrl c).

The files /var/spool/cron/cron.allow and /var/spool/cron/cron.deny and the files /var/spool/cron/at.allow and /var/spool/cron/at.deny restrict use of these commands.
These files are simply lists of user names.

Remove both files and touch cron.deny to give open access to
either *crontab* or *at*.
The *at* and *crontab* commands store scripts in /var/spool/cron/crontabs.

mail	Displays the results of the cron and at commands.

Shutting the System Down

The surest way to take control of a system is to shut it down.
UNIX file systems can be damaged when a system looses power
because the system maintains the superblocks of all file systems
in memory and it accumulates data in buffers which are written to disk
when the buffer is full.
When a system loses power,
the information in processor memory and on disk may not be the same.

#sync	Synchronizes the file systems with processor memory.

It is traditional to give two *sync* commands separated by a short interval
when ever it looks like your system is going down,
but be aware that power fluctuations could distort the information sent
doing more damage than good.
The subsequent system initialization runs *fsck* to repair the file systems.

There are better ways to stop a complex system like UNIX
that take into consideration
that users may be expecting to finish what they are doing
and that some processes are able to get their affairs in order before exitting to
avoid loosing and perhaps salvaging information.

#shutdown *hh:mm message*
#shutdown +*minutes message*

Schedule a system shutdown,
regularly announces the fact to users,
create the file /etc/nologin
five minutes prior to scheduled time

preventing external access to the system,
and put the system into a *single user* state.

#shutdown now *message*

Announces an immediate shutdown,
creates the file /etc/nologin,
prevents external access to the system,
and put the system into a *single-user* state.

Once the superuser is the only person on the system,
the system software can be serviced,
the system can be stopped and started immediately (rebooted), or
the system can be completely stopped.
The *shutdown* command is also available to users in the *operator* group.

#shutdown -r message
#reboot

Reboots the system immediately.

#reboot -- -s — Reboots the system into a single-user state.

#shutdown -h *message*
#halt

Stops the system immediately.

#shutdown -f *message*
#fastboot

Creates the file /fastboot and
reboots the system
without checking the file systems.

#fastboot -- -s — Creates the file /fastboot and
reboots the system into a single-user state
without checking the file systems.

#fasthalt — Creates the file /fastboot and
halts the system so that a subsequent boot
does not check the file systems.

#shutdown -k *message*

Simulates (kids about) a shutdown to motivate users to leave the system without halting the system.

Shutdown places the system in a single-user state by signalling the process *init*.

#kill -TERM 1	Sends the system into a single-user state.
CTRL D	Returns the system to multiuser state.

When the console does not respond to the keyboard, it may be possible to throw the system into the control of the *eeprom* monitor

Stop a	Places the system in monitor mode (>) from the console.
c	Continues the interrupted program.
b	Reboots the system.
b ?	Lists the possible devices to boot from.
b -s	Reboots in single-user mode.
k 0	Resets all peripherals.
k 1	Resets all software settings.
k 2	Resets all hardware.
n	Provides access to a set of diagnostic commands.
help *topic*	Describes commands under a particular topic.
old-mode	Returns the monitor to boot mode.