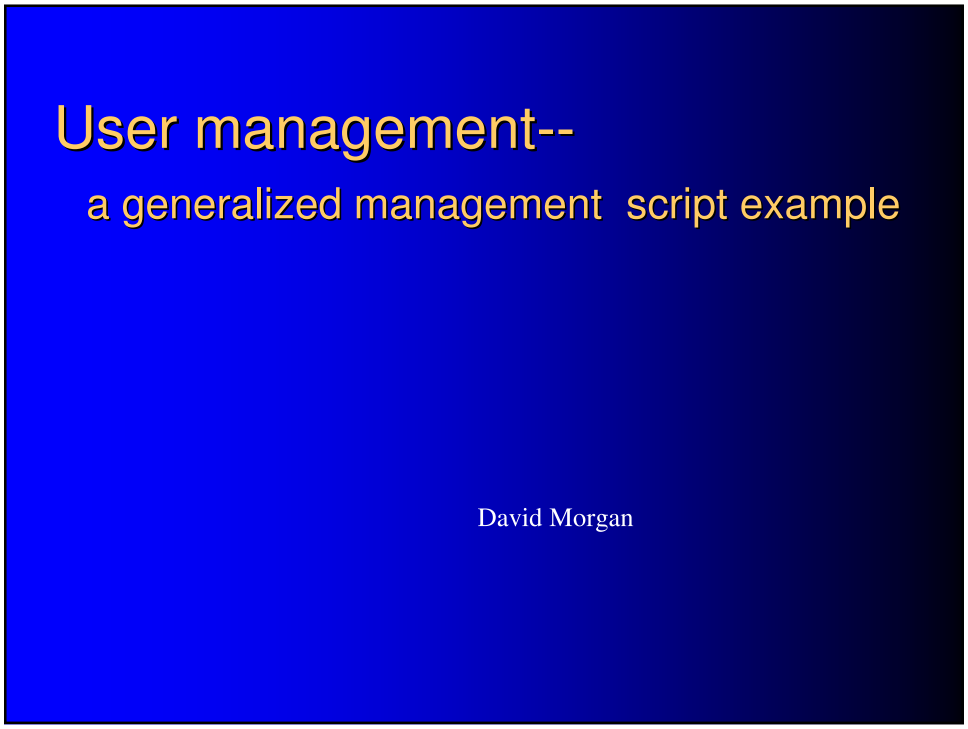
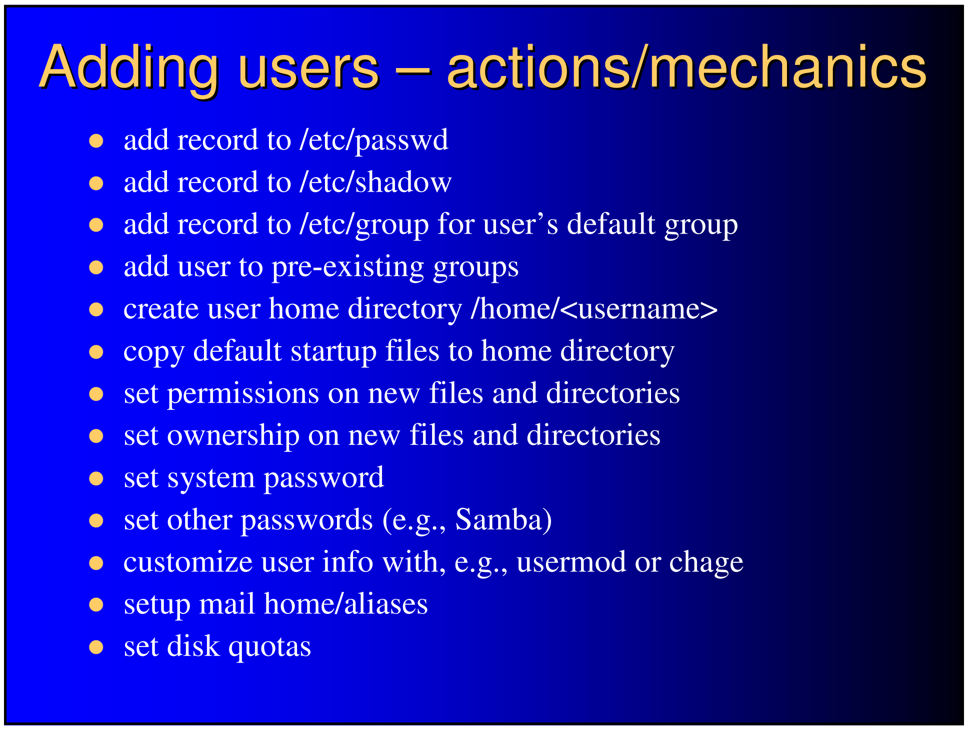


User management--

a generalized management script example

David Morgan



Adding users – actions/mechanics

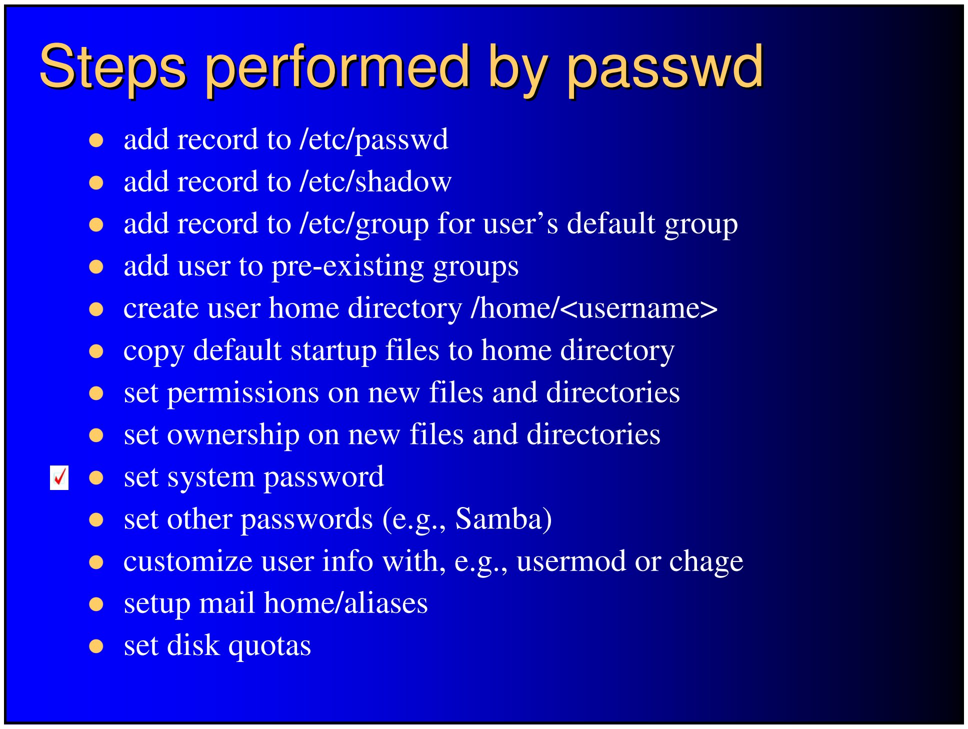
- add record to /etc/passwd
- add record to /etc/shadow
- add record to /etc/group for user's default group
- add user to pre-existing groups
- create user home directory /home/<username>
- copy default startup files to home directory
- set permissions on new files and directories
- set ownership on new files and directories
- set system password
- set other passwords (e.g., Samba)
- customize user info with, e.g., usermod or chage
- setup mail home/aliases
- set disk quotas

Process of adding users

- surprisingly extensive, isn’t it!?
- varies among sites (local policies differ)
- no utility does it all
  - some do it partially (useradd, passwd)

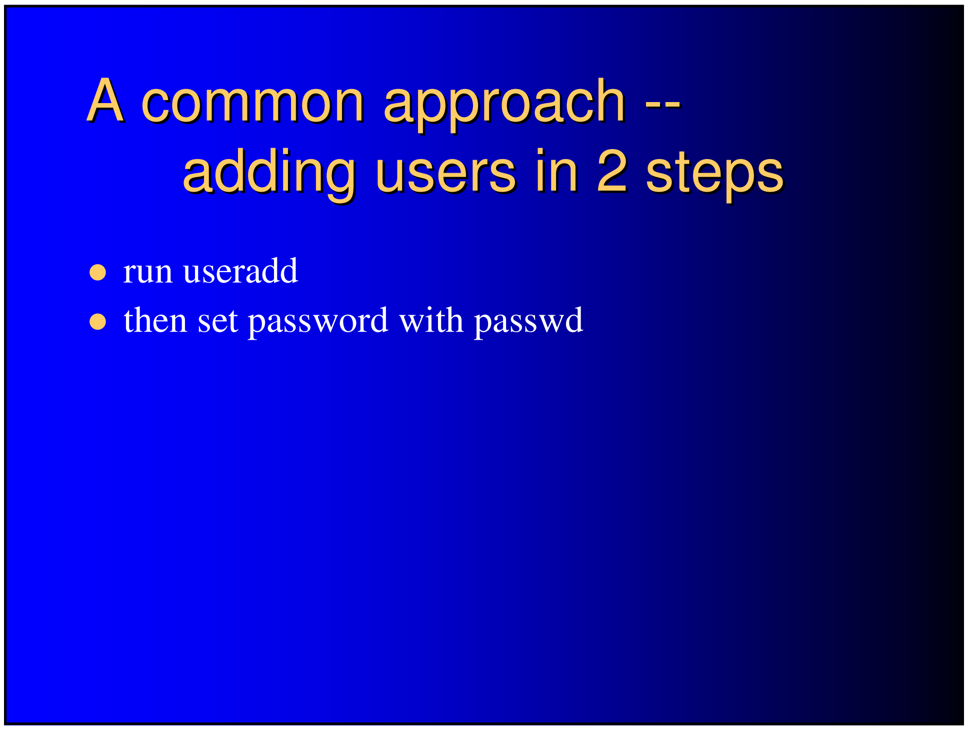
Steps performed by useradd

- ✓ add record to /etc/passwd
- ✓ add record to /etc/shadow
- ✓ add record to /etc/group for user’s default group
- add user to pre-existing groups
- ✓ create user home directory /home/<username>
- ✓ copy default startup files to home directory
- ✓ set permissions on new files and directories
- ✓ set ownership on new files and directories
- set system password
- set other passwords (e.g., Samba)
- customize user info with, e.g., usermod or chage
- setup mail home/aliases
- set disk quotas



Steps performed by passwd

- add record to /etc/passwd
- add record to /etc/shadow
- add record to /etc/group for user’s default group
- add user to pre-existing groups
- create user home directory /home/<username>
- copy default startup files to home directory
- set permissions on new files and directories
- set ownership on new files and directories
- ✓ set system password
- set other passwords (e.g., Samba)
- customize user info with, e.g., usermod or chage
- setup mail home/aliases
- set disk quotas



A common approach -- adding users in 2 steps

- run useradd
- then set password with passwd

Other approaches

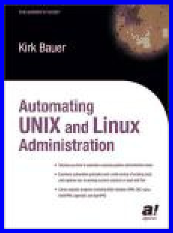
- manual - perform individual steps separately
- hybrid - some with utilities, others manually
- automated - all by script(s) you write

“Although the uaseradd and userdel commands are convenient, they are usually not sufficient to implement all of a site’s local policies. Don’t hesitate to write your own adduser and rmuser scripts; most larger sites do. …Your homebrew scripts can call the standard utilities to accomplish part of their work.”
Linux Administration Handbook Nemeth, Snyder, and Hein

Where credit is due

Following approach and scripts are from:

Automating Unix and Linux Administration,
Kirk Bauer, Apress, 2003

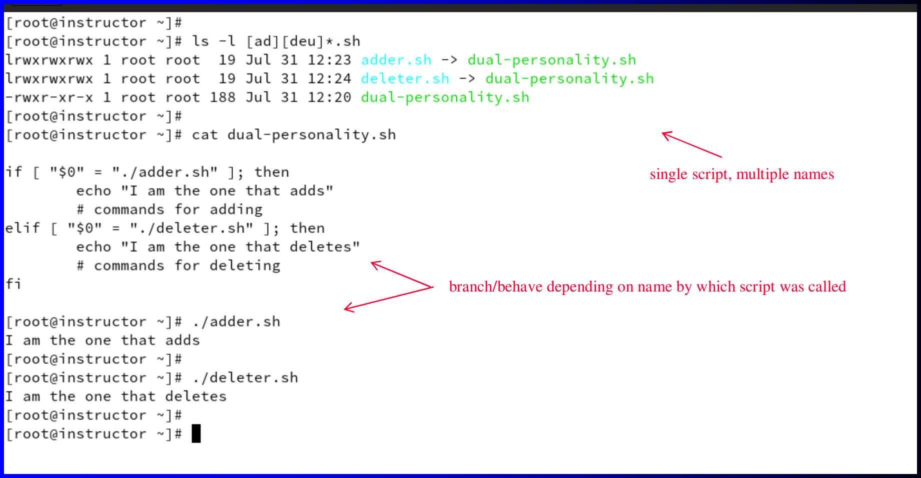


“For a small number of systems, the standard account management tools provided with your operating system are usually adequate.”
Kirk Bauer

Account mgmt script's techniques

- $0 for branching differently if called differently
- export for variable transmissibility to child
- eval

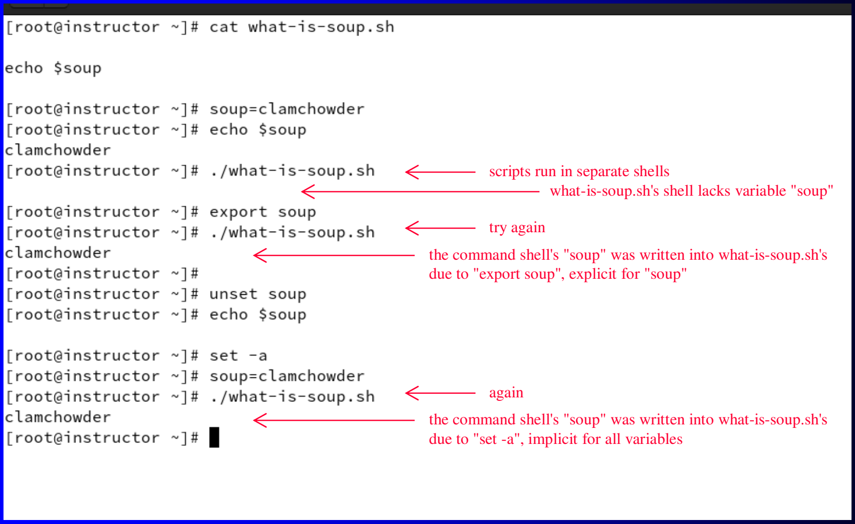
$0 - command token



```
[root@instructor ~]#
[root@instructor ~]# ls -l [ad][deu]*.sh
lrwxrwxrwx 1 root root  19 Jul 31 12:23 adder.sh -> dual-personality.sh
lrwxrwxrwx 1 root root  19 Jul 31 12:24 deleter.sh -> dual-personality.sh
-rwxr-xr-x 1 root root 188 Jul 31 12:20 dual-personality.sh
[root@instructor ~]#
[root@instructor ~]# cat dual-personality.sh

if [ "$0" = "./adder.sh" ]; then
        echo "I am the one that adds"
        # commands for adding
elif [ "$0" = "./deleter.sh" ]; then
        echo "I am the one that deletes"
        # commands for deleting
fi

[root@instructor ~]# ./adder.sh
I am the one that adds
[root@instructor ~]#
[root@instructor ~]# ./deleter.sh
I am the one that deletes
[root@instructor ~]#
[root@instructor ~]# 
```

single script, multiple names
branch/behave depending on name by which script was called

export - publish var into child



```
[root@instructor ~]# cat what-is-soup.sh

echo $soup

[root@instructor ~]# soup=clamchowder
[root@instructor ~]# echo $soup
clamchowder
[root@instructor ~]# ./what-is-soup.sh

[root@instructor ~]# export soup
[root@instructor ~]# ./what-is-soup.sh
clamchowder
[root@instructor ~]#
[root@instructor ~]# unset soup
[root@instructor ~]# echo $soup

[root@instructor ~]# set -a
[root@instructor ~]# soup=clamchowder
[root@instructor ~]# ./what-is-soup.sh
clamchowder
[root@instructor ~]#
```

scripts run in separate shells

what-is-soup.sh's shell lacks variable "soup"

try again

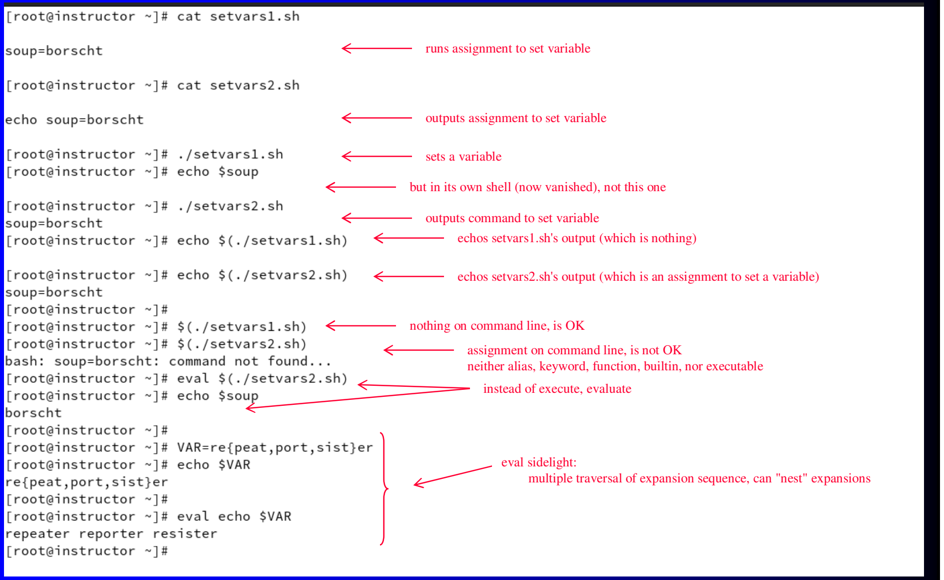
the command shell's "soup" was written into what-is-soup.sh's due to "export soup", explicit for "soup"

again

the command shell's "soup" was written into what-is-soup.sh's due to "set -a", implicit for all variables

eval

```
eval [arg ...]
       The  args  are  read  and concatenated together into a single command.  This command is then read and executed by the
       shell, and its exit status is returned as the value of eval.  If there are no args,  or  only  null  arguments,  eval
       returns 0.
```



```
[root@instructor ~]# cat setvars1.sh

soup=borscht

[root@instructor ~]# cat setvars2.sh

echo soup=borscht

[root@instructor ~]# ./setvars1.sh
[root@instructor ~]# echo $soup

[root@instructor ~]# ./setvars2.sh
soup=borscht
[root@instructor ~]# echo $(./setvars1.sh)

[root@instructor ~]# echo $(./setvars2.sh)
soup=borscht
[root@instructor ~]#
[root@instructor ~]# $(./setvars1.sh)
[root@instructor ~]# $(./setvars2.sh)
bash: soup=borscht: command not found...
[root@instructor ~]# eval $(./setvars2.sh)
[root@instructor ~]# echo $soup
borscht
[root@instructor ~]#
[root@instructor ~]# VAR=re{peat,port,sist}er
[root@instructor ~]# echo $VAR
re{peat,port,sist}er
[root@instructor ~]#
[root@instructor ~]# eval echo $VAR
repeater reporter resister
[root@instructor ~]#
```

runs assignment to set variable

outputs assignment to set variable

sets a variable

but in its own shell (now vanished), not this one

outputs command to set variable

echos setvars1.sh's output (which is nothing)

echos setvars2.sh's output (which is an assignment to set a variable)

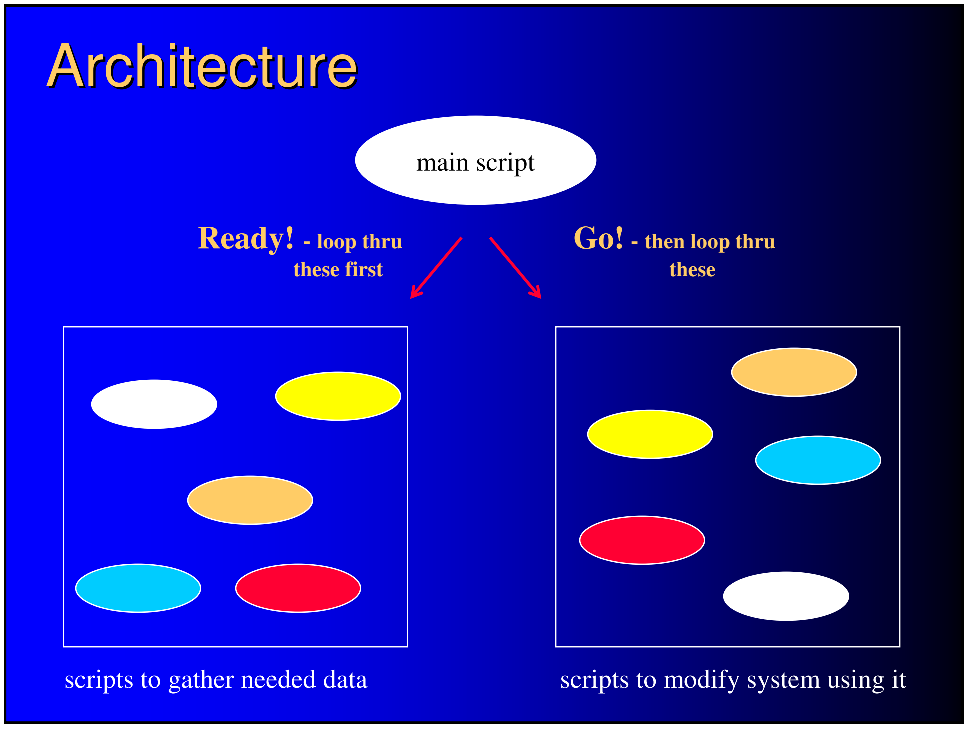
nothing on command line, is OK

assignment on command line, is not OK
neither alias, keyword, function, builtin, nor executable

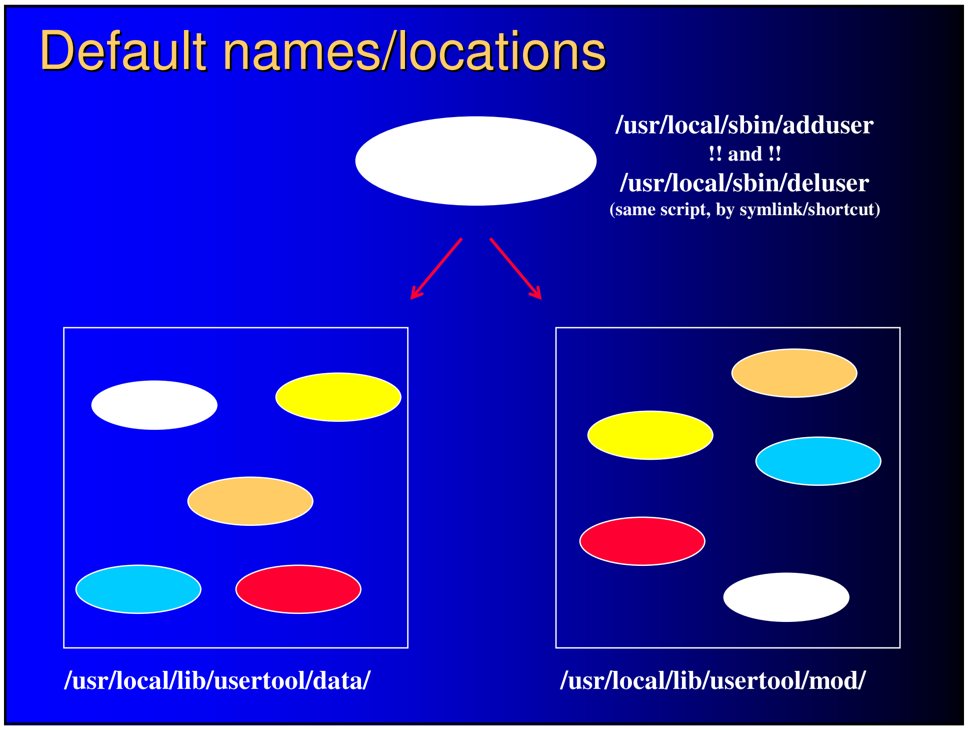
instead of execute, evaluate

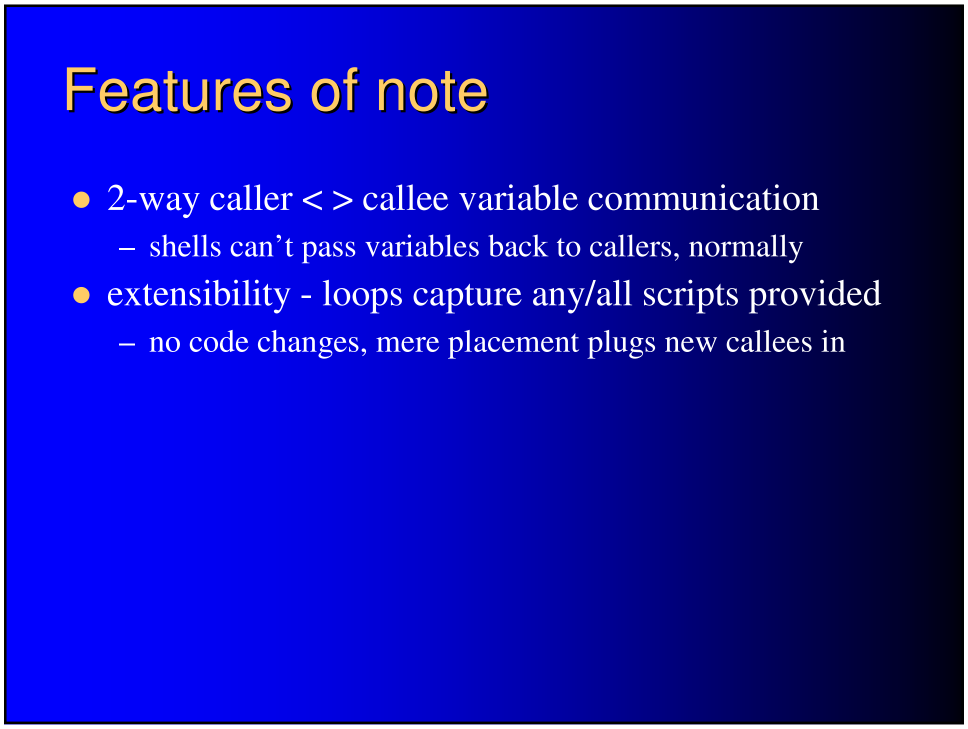
eval sidelight:
multiple traversal of expansion sequence, can "nest" expansions

Architecture


main script
Ready! - loop thru these first
Go! - then loop thru these
scripts to gather needed data
scripts to modify system using it

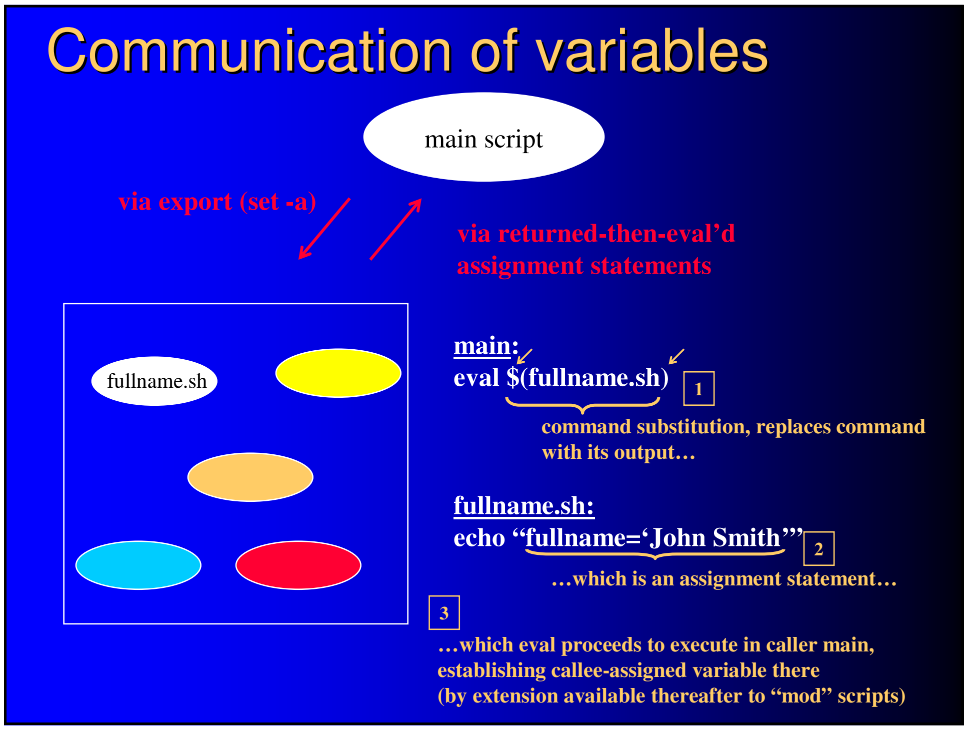
Default names/locations


/usr/local/sbin/adduser
!! and !!
/usr/local/sbin/deluser
(same script, by symlink/shortcut)
/usr/local/lib/usertool/data/
/usr/local/lib/usertool/mod/



Features of note

- 2-way caller < > callee variable communication
  - shells can’t pass variables back to callers, normally
- extensibility - loops capture any/all scripts provided
  - no code changes, mere placement plugs new callees in



Communication of variables

main:
eval $(fullname.sh) $\boxed{1}$

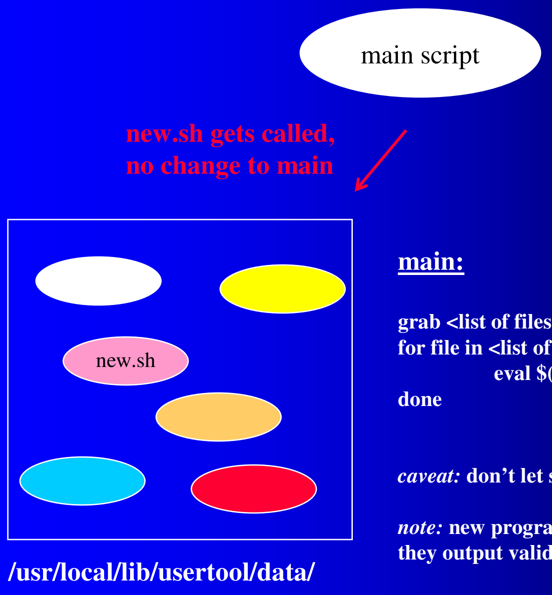
command substitution, replaces command with its output…

fullname.sh:
echo “fullname=‘John Smith’” $\boxed{2}$

…which is an assignment statement…

$\boxed{3}$
…which eval proceeds to execute in caller main, establishing callee-assigned variable there (by extension available thereafter to “mod” scripts)

Auto-extensibile by drop-in



main:

```
grab <list of files> in /usr/local/usertool/data/
for file in <list of files> ; do
        eval $( $file )   # execute the output of the file
done
```

caveat: **don't let stray files in the directory!**

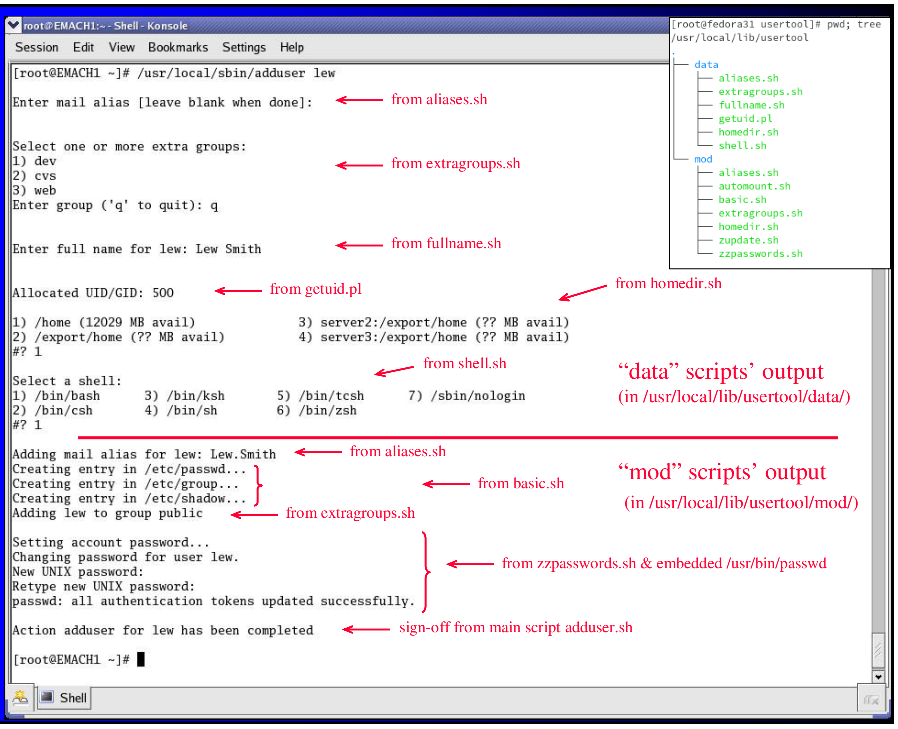
note: **new programs in any language OK, as long as they output valid shell commands on stdout**

Example data flows

fullname.sh shell.sh →	basic.sh →	/etc/passwd
extaragroups.sh →	extaragroups.sh →	/etc/group
homedir.sh →	homedir.sh →	/home or …

e.g., fullname.sh and shell.sh in the data directory solicit from user his full name and choice of shell, pass results to basic.sh in the mod directory, which from that and other info composes a standard user record and inscribes it into the "official user database" /etc/passwd

Typical usage screenshot

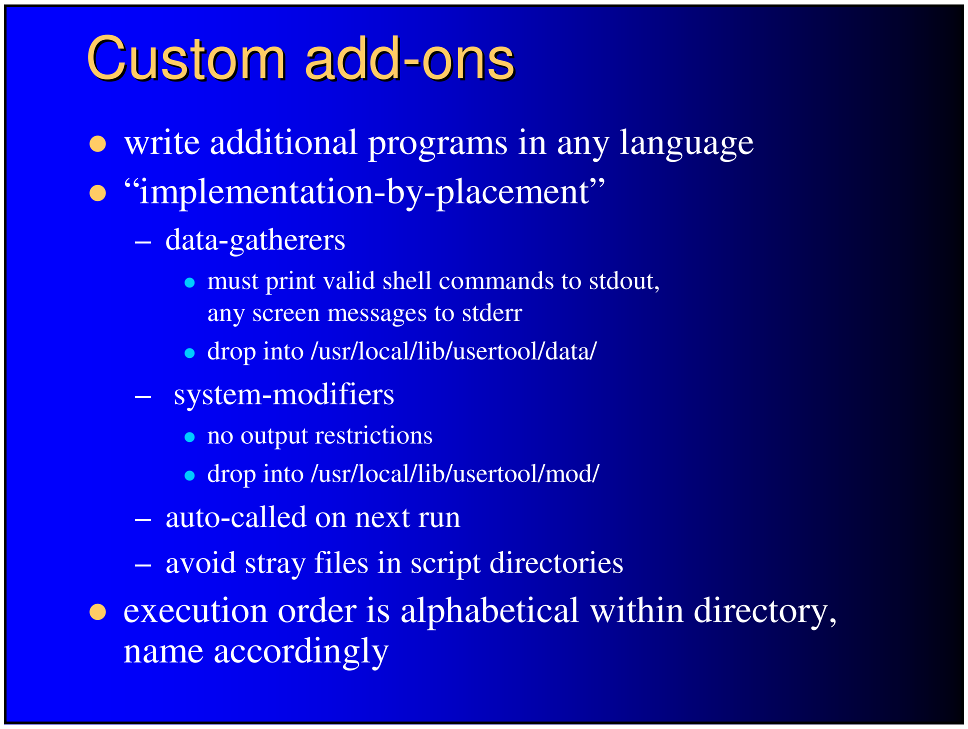

root@EMACH1:~ - Shell - Konsole
Session Edit View Bookmarks Settings Help
Shell
from aliases.sh
from extragroups.sh
from fullname.sh
from getuid.pl
from homedir.sh
from shell.sh
"data" scripts' output
(in /usr/local/lib/usertool/data/)
from aliases.sh
from basic.sh
from extragroups.sh
"mod" scripts' output
(in /usr/local/lib/usertool/mod/)
from zzpasswords.sh & embedded /usr/bin/passwd
sign-off from main script adduser.sh

```
[root@EMACH1 ~]# /usr/local/sbin/adduser lew

Enter mail alias [leave blank when done]:


Select one or more extra groups:
1) dev
2) cvs
3) web
Enter group ('q' to quit): q


Enter full name for lew: Lew Smith


Allocated UID/GID: 500

1) /home (12029 MB avail)             3) server2:/export/home (?? MB avail)
2) /export/home (?? MB avail)         4) server3:/export/home (?? MB avail)
#? 1

Select a shell:
1) /bin/bash      3) /bin/ksh       5) /bin/tcsh      7) /sbin/nologin
2) /bin/csh       4) /bin/sh        6) /bin/zsh
#? 1

Adding mail alias for lew: Lew.Smith
Creating entry in /etc/passwd...
Creating entry in /etc/group...
Creating entry in /etc/shadow...
Adding lew to group public

Setting account password...
Changing password for user lew.
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.

Action adduser for lew has been completed

[root@EMACH1 ~]#
```

```
[root@fedora31 usertool]# pwd; tree
/usr/local/lib/usertool
.
├── data
│   ├── aliases.sh
│   ├── extragroups.sh
│   ├── fullname.sh
│   ├── getuid.pl
│   ├── homedir.sh
│   └── shell.sh
└── mod
    ├── aliases.sh
    ├── automount.sh
    ├── basic.sh
    ├── extragroups.sh
    ├── homedir.sh
    ├── zupdate.sh
    └── zzpasswords.sh
```

deluser alternative functionality

- "adduser" callable by alternative name "deluser"
- it checks by which name it was called
- undoes most (not all) of its "adduser" actions when called as "deluser" instead
- undoing remainder can be implemented as a custom add-on script you supply



Custom add-ons

- write additional programs in any language
- “implementation-by-placement”
  - data-gatherers
    - must print valid shell commands to stdout, any screen messages to stderr
    - drop into /usr/local/lib/usertool/data/
  - system-modifiers
    - no output restrictions
    - drop into /usr/local/lib/usertool/mod/
  - auto-called on next run
  - avoid stray files in script directories
- execution order is alphabetical within directory, name accordingly