Setting up your environment

- Environment variables – these variables are passed to child processes
- Aliases – modify the meaning of "commands"
- History – a record of your shell commands
- Command completion – lets you save keystrokes

Environmental variables

- Environmental variables are passed to child processes at invocation. (The child process can of course ignore them if it likes.)
- Children cannot modify parent's environmental variables – any modification by a child process are local to the child and any children it might create.

Environmental variables

☞ The traditional C "main" is usually defined something like:

```
int main(int argc, char *argv[], char *envp[])
```

Setting environmental variables

CSH/TCSH: `setenv VARIABLE VALUE`

BASH: `export VARIABLE=VALUE`

old SH: `VARIABLE=VALUE ; export VARIABLE`

Note: there are a few special variables such as `path` and `home` that CSH/TCSH autosynchronizes between the two values.

Setting environmental variables

```
[langley@sophie 2006-Fall]$export VAR1=value
[langley@sophie 2006-Fall]$ bash
[langley@sophie 2006-Fall]$ echo $VAR1
value
[langley@sophie 2006-Fall]$ exit
exit
[langley@sophie 2006-Fall]$ csh
[langley@sophie 2006-Fall]$ echo $VAR1
value
```

Setting environmental variables

```
[langley@sophie 2006-Fall]$ csh
[langley@sophie 2006-Fall]$ setenv VAR2 bigvalue
[langley@sophie 2006-Fall]$ csh
[langley@sophie 2006-Fall]$ echo $VAR2
bigvalue
[langley@sophie 2006-Fall]$ exit
[langley@sophie 2006-Fall]$ exit
[langley@sophie 2006-Fall]$ bash
[langley@sophie 2006-Fall]$ echo $VAR2
bigvalue
```

Unsetting environmental variables

CSH/TCSH: `unsetenv VAR`

SH/BASH: `unset VAR`

You can also leave it as local variable in bask with `export -n VAR`.

Unsetting environmental variables

```
[langley@sophie 2006-Fall]$ csh
[langley@sophie 2006-Fall]$ setenv VAR99 testvar
[langley@sophie 2006-Fall]$ csh
[langley@sophie 2006-Fall]$ echo $VAR99
testvar
[langley@sophie 2006-Fall]$ unsetenv VAR99
[langley@sophie 2006-Fall]$ echo $VAR99
VAR99: Undefined variable.
[langley@sophie 2006-Fall]$ exit
[langley@sophie 2006-Fall]$ exit
[langley@sophie 2006-Fall]$ echo $VAR99
testvar
```

Unsetting environmental variables

```
[langley@sophie 2006-Fall]$ export VAR50=test
[langley@sophie 2006-Fall]$ bash
[langley@sophie 2006-Fall]$ echo $VAR50
test
[langley@sophie 2006-Fall]$ unset VAR50
[langley@sophie 2006-Fall]$ echo $VAR50

[langley@sophie 2006-Fall]$ exit
exit
[langley@sophie 2006-Fall]$ echo $VAR50
test
[langley@sophie 2006-Fall]$ export -n VAR50
[langley@sophie 2006-Fall]$ echo $VAR50
test
[langley@sophie 2006-Fall]$ bash
```

```
[langley@sophie 2006-Fall]$ echo $VAR50
```

Displaying your environment

BASH: `env`, `printenv`, `set`, `declare -x`, `typeset -x`

CSH: `env`, `printenv`, `setenv`

Predefined environmental variables

What is "predefined" is not so much the value of the variable as its name and its normal use.

- `PATH` : a list of directories to visit. They are delimited with ":". Note that csh/tcsh "autosynchronize" this variable.

- `EDITOR` : the default editor to start when you run a program that involves editing a file, such as `crontab -e`.

- `PRINTER` : the default printer to send to.
- `PWD` : your present working directory.
- `HOME` : your home directory.
- `SHELL` : the path to your current shell. (Be cautious with this one: in some shells, it is instead `shell`).
- `USER` : your username.
- `TERM` : your terminal type.
- `DISPLAY` : used by programs to find the X server to

display their windows.

Aliases

An `alias` allows you to abbreviate a command. For instance, instead of using `/bin/ls -al`, you might abbreviate it to `ll` with:

SH/BASH: `alias ll=‘‘/bin/ls -al’’`

CSH/TCSH: `alias ll ‘‘/bin/ls -al’’`

Removing aliases

You can remove an alias with `unalias`.

Example:

```
unalias ll
```

`which, whatis, whereis, locate`

The program (or built-in) `which` simply gives you the path to the named executable as it would be interpreted by your shell invoking that executable, and is created by examining your path.

The program `locate` looks in a database for all accessible files in the filesystem that contain the substring you specify. You can also specify a regular expression, such as

```
locate -r 'ab.*ls'
```

The program `whatis` will give you the description line from the `man` page for the command you specify. (N.B.: You can also search the `man` page descriptions with `man -k keyword`.)

The program `whereis` will give you both the path to the executable named and the page to its manpage.

Setting your prompt

SH/BASH: `PS1='% '`

CSH/TCSH: `set prompt='% '`

"Sourcing" commands

Because ordinarily running a shell script means first forking a child process and then exec-ing the script in that child shell, it is not possible to modify the current shell's environmental variables from just running a script.

Instead, we do what is called "sourcing" the script, which means simply executing its commands (such as setting environmental variables) inside the current shell process.

CSH/TCSH: `source FILE`

SH/BASH: `. FILE`

N.B.: modern versions of bash also support the `source` built-in.

.login , .profile

When you login, your user shell is started with `-l`. For sh/bash, this means that shell will source your `.profile` file; for csh/tcsh, this means sourcing your `.login` file.

Typically, you would want your environmental variables in that file, and any other one-time commands that you want to do when logging in, such as checking for new email.

Shell `.*rc` files

For each shell that you start, generally a series of "run command" files, abbreviated as "rc" will be sourced. In these you can set up aliases and variables that you want for every shell (including those that are not interactive, such as those running under a crontab.)

BASH: `.bashrc`

CSH: `.cshrc`

There is also a `.tschrc` for `tcsh`. History, `sh` did not

look for configuration files except when invoked as a login shell.

`.*rc` files in general

In general, many program use .*rc files. Some will ask you to setup the file; some will create it for you. Some want a whole directory.

- `.gvimrc`
- `.procmailrc`
- `.gtkrc`
- `.xfigrc`

☞ `.acrorc`

.gvimrc

- Set the background
- Set the size and type of the font
- Set the size of the window in characters
- Turn on or off syntax highlighting

.procmailrc

The syntax is quite obscure, but you can apply arbitrary rules to your incoming email via your `.procmailrc` file.

.procmailrc example

```
DOMAIN="<$1>"
RECIPIENT="<$2>"
WHATSIT="<$3>"
VERBOSE=on
LOGFILE=/tmp/procmail2.log
LOGABSTRACT=all
ROOTHOMEDIR=/home/vmail-users
ROOTINBOXDIR=/var/spool/vbox

:0
* RECIPIENT ?? ()\/[^<]*@
* MATCH ?? ()\/.*[^@]
{
   USER=$MATCH
}
```

```
:0 a
* DOMAIN ?? ()\/[^<].*[^>]
{
   DOMAINNOBRACKET=$MATCH
}

:0 a
${ROOTINBOXDIR}/${DOMAINNOBRACKET}/${USER}
```

Shell history

You can modify the number of lines kept in your history:

bash: `HISTSIZE=SOMENUMBER`

csh/tcsh: `set history=SOMENUMBER`

Your shell history lets you do many things: search commands that you ran in the past, re-execute commands, modify them, or save them off (bash lets you do the latter automatically in your `.bash_history` file.)

Command history substitution

- `!!` → repeat last command
- `â^b̂` → repeat last command, but change `a` to `b`
- `!-N` → repeat the command N back in your history
- `history` → display the history
- `history N` → display the last N lines of history
- `!N` → repeat command N

☞ `!STRING` $\longrightarrow$ repeat the last command that started with STRING.

Using previous command arguments

- `!$` → refers to the last argument of the previous command
- `!caret` → refers to the first argument of the previous command
- `!*` → refers to the all of the arguments of the previous command
- `!:n*` → refers to the arguments N through the last

argument of the previous command

Command line manipulation

You can use the arrow keys to move through your history, and back and forth on command lines.

With bash, you can use the default emacs key-bindings for thing such as end-of-line (ctrl-e) and beginning-of-line (ctrl-b).

Complete word function

If you are in the first word of a command, you can find all the matching commands up to that point with a TAB character.

If you are else in the line, you can use the TAB character to show all matching filenames in the current working directory, or if you have started an absolute path, then matching items down the path.