**~CHAPTER 3: Customizing Your Environment~**

**1. Understanding Environment**

* An environment is a collection of tools and concepts designed for usability and comfort.
* Example: A work desk environment involves tools like pens, phone, calculator, etc., organized for efficiency.

**2. UNIX Shell as an Environment**

* The UNIX shell provides a conceptual environment with elements like files, directories, standard input/output.
* UNIX tools include file manipulation commands, text editors, and print queues.
* Customization involves directory setup, file naming, and keyboard/display configuration.

**3. Importance of Customization**

* Customizing makes the environment more productive and tailored to personal needs.
* Similar to arranging a physical desk, a customized shell boosts efficiency.

**4. Four Key Bash Customization Features**

* Special Files:
  + .bash\_profile, .bash\_logout, .bashrc
  + Used during login, logout, or shell startup to set up environment.
* Aliases:
  + Create shortcuts for commands or command sequences for convenience.
* Options:
  + Enable or disable various environment behaviors.
* Variables:
  + Named values that affect shell/program behavior; customizable and dynamic.

**5. Foundation for Advanced Customization**

* These features form the basis for more complex shell programming and are common across most UNIX shells.

***The .bash\_profile, .bash\_logout, and .bashrc Files***

### **1. Purpose of Special Bash Files**

* Bash reads specific files in your **home directory** to customize the shell environment.
* These files are:
  + .bash\_profile – run at **login**
  + .bashrc – run when a **new shell** is invoked (subshell)
  + .bash\_logout – run at **logout**

### 

### **2. .bash\_profile**

* Most important startup file for login sessions.
* Used to **set environment variables** and initialize settings.
* Only **one** of the following is read: .bash\_profile, .bash\_login, or .profile (in that order).
* Commonly used to:
  + Set the PATH
  + Set shell prompts
  + Define default editor or tools

### **3. Alternative Login File Logic**

* If .bash\_profile doesn't exist, bash checks for:
  1. .bash\_login
  2. .profile

You can reuse .profile and add bash-specific logic in .bash\_profile, using:  
source .profile

### **4. .bashrc**

* Executed when a **subshell** is started (e.g., when typing bash).
* Place commands here that should run in all interactive shells, like:
  + Aliases
  + Shell options
  + Custom functions
* Can be sourced from .bash\_profile to ensure commands run in both login and non-login shells.

**5. .bash\_logout**

* Optional file.
* Runs **at logout** of a login shell.
* Useful for:
  + Deleting temp files
  + Logging session info
  + Cleanup tasks

### **6. Applying Changes Immediately**

* Changes to .bash\_profile require:
  + Re-logging in **OR**
  + Running source .bash\_profile to reapply changes without logging out.

Key Points: Aliases in Bash

1. What Are Aliases?

* Aliases are shortcuts or mnemonics for commands or command strings.
* They help simplify frequently used or complex command combinations.

2. Syntax and Definition

Defined using the format:  
alias name=command

* No spaces around the = sign.
* Can be defined:
  + On the command line
  + In .bashrc (for subshells)
  + In .bash\_profile (for login shells)

3. Common Use Cases

Mnemonics for cryptic commands:  
 alias search=grep

Fixing common typos:

alias emcas=emacs

alias mali=mail

alias gerp=grep

Shorthand for long commands:  
  
alias cdvoy='cd sipp/demo/animation/voyager'

alias lf='ls -F'

alias printall='pr \* | lpr'

4. Recursive Aliases

Aliases can reference other aliases:  
  
alias printall='pr \* | lpr'

alias pa=printall

Bash prevents infinite loops by not expanding an alias recursively if it would lead to itself:  
  
alias ls='ls -l' # Works as expected

alias listfile=ls

alias ls=listfile # Ignored to prevent loop

5. Limitation: Start of Command Only

* Aliases only work at the beginning of a command.

You can’t alias just a directory name:  
  
alias anim=sipp/demo/animation/voyager

cd anim # This won't work

6. Workaround: Trailing Space Technique

To allow alias expansion of the second word, use a trailing space:  
  
 alias cd='cd '

alias anim='sipp/demo/animation/voyager'

cd anim # Now works as expected

7. Managing Aliases

View all aliases:  
  
alias

View a specific alias:  
  
alias name

Remove an alias:  
  
unalias name

8. Limitations and Alternatives

* Aliases can't take arguments (unlike C shell aliases).

Key Points: Bash Shell Options

1. What Are Shell Options?

* Options allow users to modify the behavior of the Bash shell.
* Each option is a toggle—it can be turned on or off.
* More powerful than aliases, as they affect how the shell behaves internally.

2. Command Syntax for Setting Options

* Use set -o optionname to turn an option ON.
* Use set +o optionname to turn an option OFF.

You can set multiple options in one command by repeating -o or +o:  
 set -o option1 -o option2

set +o option1 +o option2

3. Note on Syntax: - vs +

* The use of - to turn an option on, and + to turn it off, is counterintuitive.
* This stems from UNIX conventions where - denotes a command-line flag.

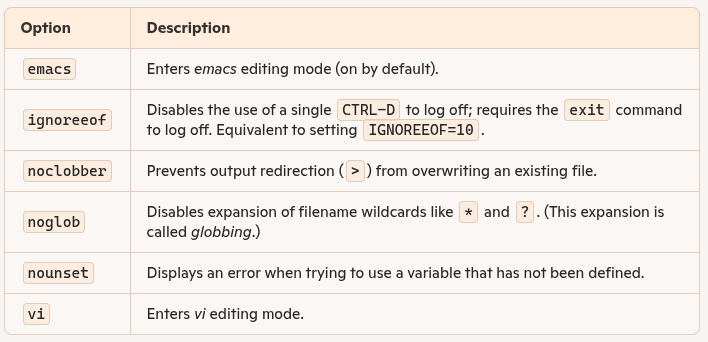
4. One-Letter Abbreviations

* Many options have single-character abbreviations inherited from the Bourne shell.
* Example:
  + set -o noglob is equivalent to set -f
* These are legacy features, and their use is generally discouraged for clarity.

5. Default State

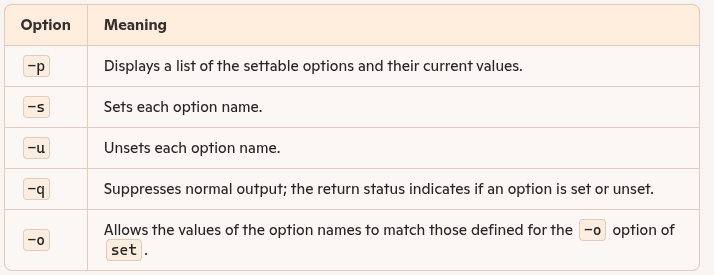
* Most options are off by default, unless otherwise noted.

6. Option Management Use Cases

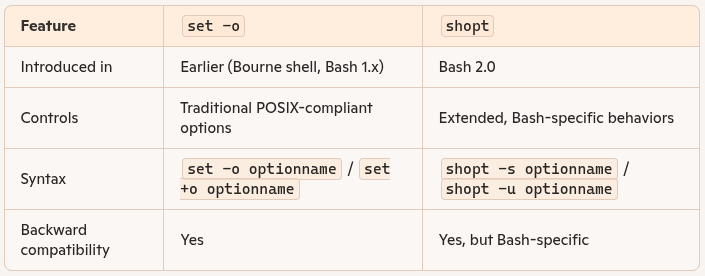
* Customize shell behavior for:
  + Filename expansion
  + Command history
  + Job control
  + Error handling, etc.
* Often placed in .bashrc or .bash\_profile for persistent effect.  
  

shopt:-efinition:

* shopt stands for "shell options".
* It is a built-in Bash command introduced in Bash 2.0.  
  Allows you to enable or disable shell behaviors that are not covered by set -o



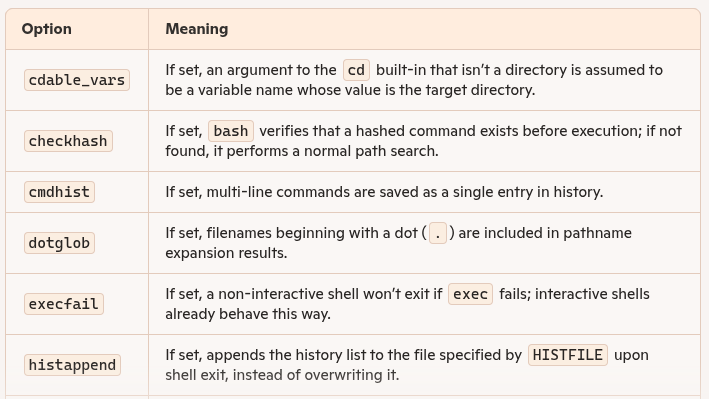
Difference between set -o and shopt:-



### **What Are Shell Variables?**

* **IN TERMINAL[SAMPLE CODE]**
* Shell variables are **named storage** locations used to store data (like strings or numbers) in your shell environment.
* Unlike aliases or options, they **store values**, not commands or toggles.
* Used for **customizing shell behavior**—e.g., prompt style, editor preference, etc.

Table 3-3. shopt option names:-





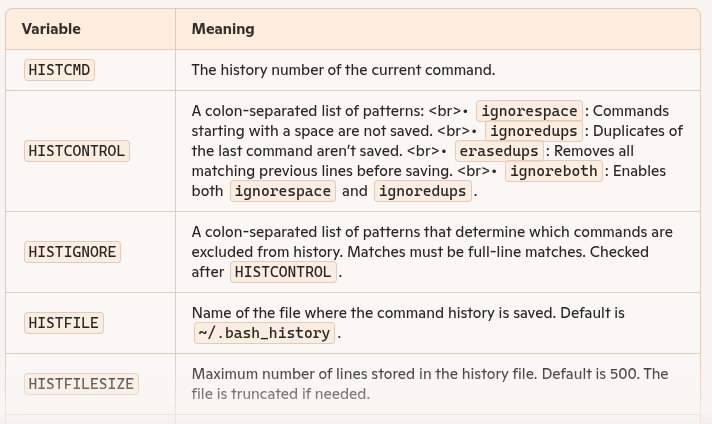
Double Quotes (" "):

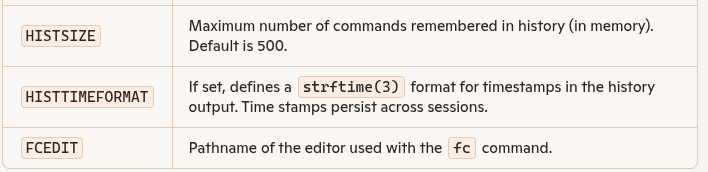
* Special characters like $ (for variables) are evaluated.
* Prevents word splitting (i.e., maintains whitespace).
* Preferred when variables are involved.

Single Quotes (' '):

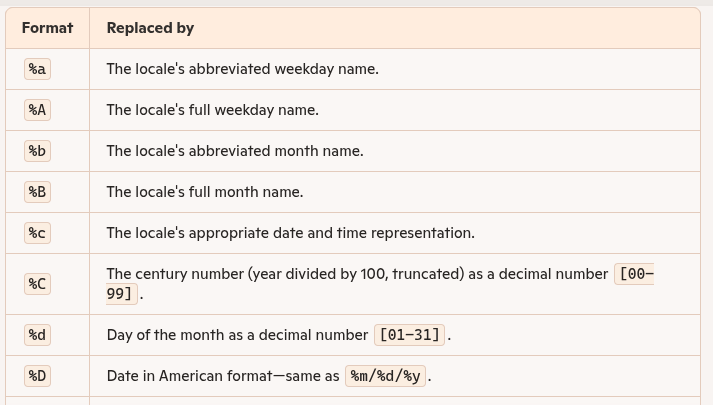
* Nothing inside is interpreted — variables are not expanded.
* Best for literal strings without variables.

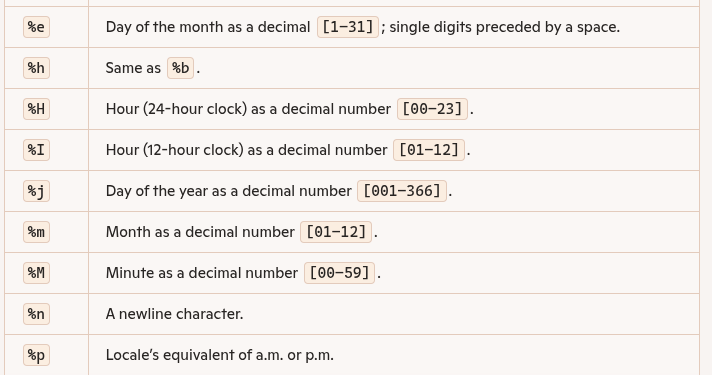
**Table 3-4. Editing mode variables**

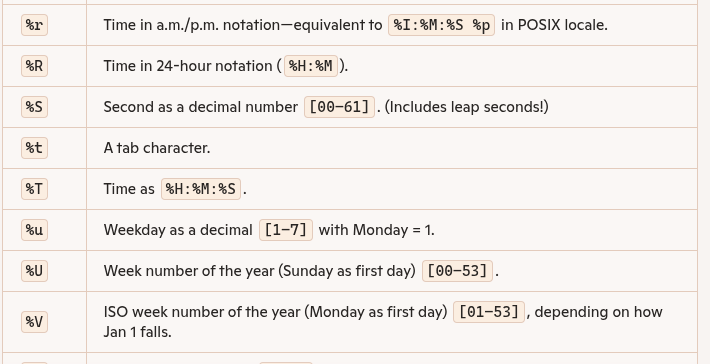
****

****

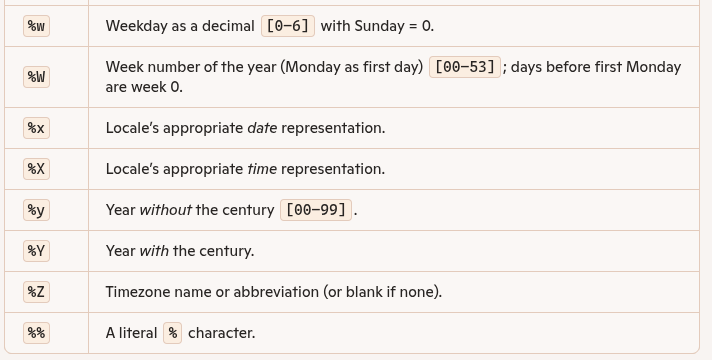
| **Topic** | **📋 Key Points** |
| --- | --- |
| **HISTCMD** | **Shows the current command number in an interactive shell. If unset, it loses its special meaning.** |
| **HISTFILESIZE** | **Sets the maximum number of commands saved in the history file. If file has more lines, it's truncated.** |
| **HISTSIZE** | **Sets the maximum number of commands kept in memory during the current session. On exit, up to this many are written to the history file.** |
| **HISTCONTROL** | **A colon-separated list controlling what gets saved to history. Options: • ignorespace – ignore commands starting with a space • ignoredups – ignore consecutive duplicate commands** |
| **HISTIGNORE** | **A pattern list to prevent certain commands from entering history. Example: l\*:& ignores commands starting with l and duplicates.** |
| **HISTTIMEFORMAT** | **Adds a timestamp to each command in history. If unset or empty, no timestamps are added.** |
| **Timestamps Format** | **Uses standard strftime format (e.g., %F %T for date and time). Output may depend on system locale.**  **Table 3-5. Time stamp formats** |







| **Concept** | **Details** |
| --- | --- |
| **CDPATH Variable** | Similar to PATH, CDPATH is a colon-separated list of directories that the cd command will search. |
| **Default Behavior** | If CDPATH is not set, cd dirname searches **only in the current directory**. |
| **Purpose** | Allows quick navigation to frequently used directories without typing full paths. |
| **Example Setting** | CDPATH=:~/sipp/demo/animation/voyager means: look in current directory (:) and in the specified path. |
| **Behavior** | If cd doc is entered and doc doesn't exist in the current directory, the shell searches ~/sipp/demo/animation/voyager/doc. |
| **Usage Tip** | Useful for projects involving multiple subdirectories (like src, bin, doc). Update CDPATH as project paths change. |
| **cdable\_vars Option** | If enabled (shopt -s cdable\_vars), any non-directory argument to cd is treated as a **variable name**. |
| **Example Use** | Define anim=~/sipp/demo/animation/voyager, then cd anim changes directory to that path (if cdable\_vars is on). |
| **Advantage** | Saves time and typing when navigating long or deeply nested paths. |
| **Enabling cdable\_vars** | Run: shopt -s cdable\_vars to activate variable-based directory jumping. |



### 

### 

### **Definition: Mail Variables in Bash**

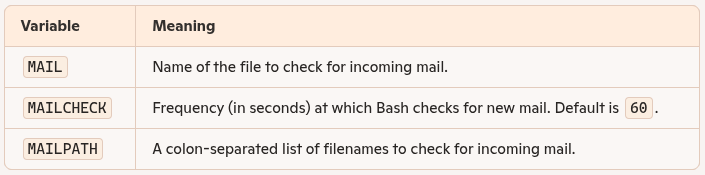
**Mail variables** are special **shell variables** in bash that allow the shell to notify the user of **new mail messages**.

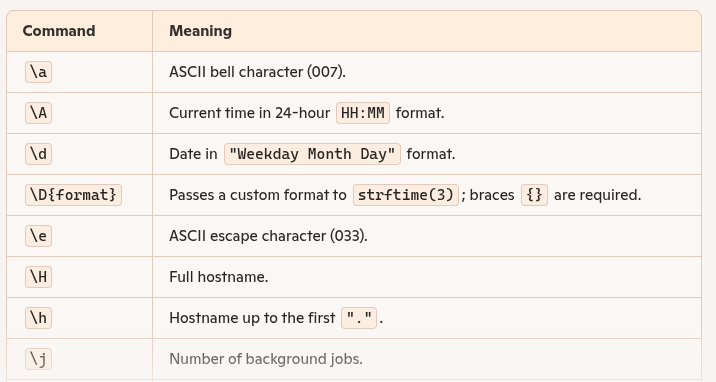
Since the **mail program doesn't run continuously**, it can't alert you directly when new mail arrives. Instead, **bash periodically checks your mail file** to see if it has been modified (i.e., if new mail has arrived) and displays a notification accordingly.

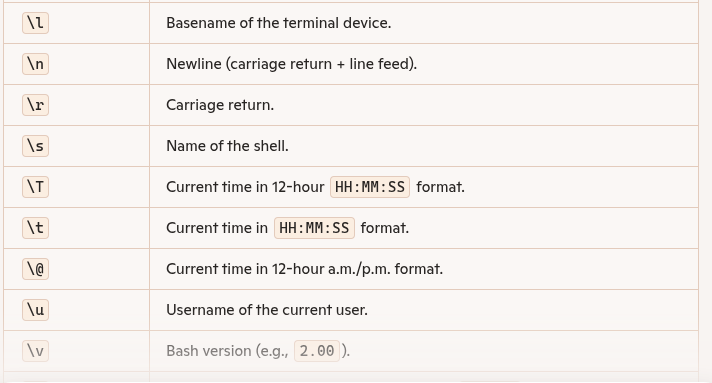
These variables control **how, when, and where** bash checks for mail updates, including:

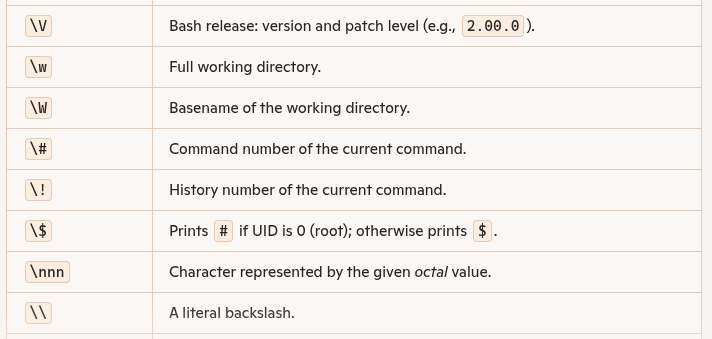
* The **location of the mail file**.
* The **frequency** of checks.
* The **message** displayed upon detecting new mail.

Table 3-6. Mail variables





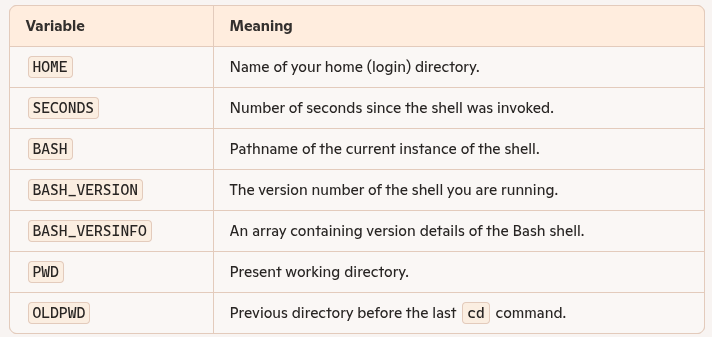




### **Definition: Miscellaneous Variables in Bash**

Miscellaneous variables are shell variables used for various auxiliary tasks, such as indicating status, configuration, or environment settings. Unlike customization-focused variables (like PS1 for the prompt or HISTSIZE for history), these variables often provide system or shell state information, or control lower-level behaviors.

### **Key Characteristics:**

* Serve **special purposes** in the shell.
* Typically **read-only** or system-assigned.
* Some track **return status** ($?), **last argument** ($\_), etc.
* Help in **scripting**, **debugging**, or **advanced shell use**.
* Listed for reference, not usually modified by users.

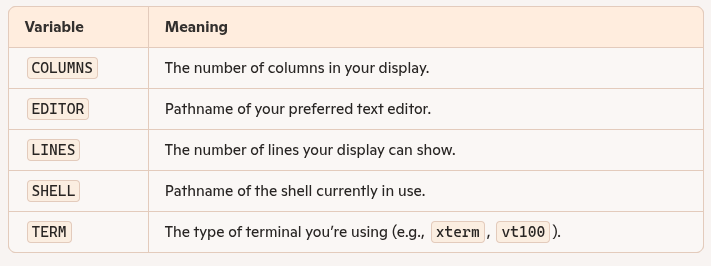
Key Points: Customization and Subprocesses

1. **Shell Customization Scope**:
   * Some shell variables affect **only the shell itself**, while others are used by **external commands** or subprocesses.
2. **Subprocess Definition**:
   * Every command run in the shell is executed in a **subprocess** (a child process created by the shell).
3. **Environment Visibility**:
   * Only a **subset of shell settings** (like exported environment variables and certain options) are **visible to subprocesses**.
   * Subprocesses **cannot modify** the parent shell’s settings.
4. **One-way Information Flow**:
   * Information can flow **from parent shell to subprocess**, but **not the other way around**.
5. **bash Programs vs. Interactive Shells**:
   * **bash scripts/programs** as subprocesses can access almost all exported options and variables.
   * **Interactive shells** might behave differently based on the shell’s context and how variables/options are passed.

**Important Key Points: Environment Variables**

1. Definition:
   * Environment variables are a special kind of shell variable that is visible to subprocesses.
2. Common Built-in Environment Variables:
   * Examples: HOME, MAIL, PATH, PWD, TERM, EDITOR.
3. Purpose:
   * They let subprocesses (like text editors or mail clients) know about your environment (e.g., which editor to use, terminal type, etc.).
4. Creating Environment Variables:
   * First define a variable, then use export to make it part of the environment.
5. Combined Assignment and Export:
   * You can assign and export in one line using: export VAR=value.
6. Temporary Environment for a Single Command:
   * You can set an environment variable just for the duration of one command using VAR=value command.
7. Preservation of Parent Shell Environment:
   * When using VAR=value command, the original shell's variable value remains unchanged.
8. Viewing Environment Variables:
   * Use export or export -p to list all current environment variables.
9. Common in .bash\_profile:
   * Environment variables are frequently defined in .bash\_profile for persistent setup on login shells.
10. Widely Used Standard Variables:
    * Variables like PATH, SHELL, EDITOR, and MANPATH are commonly used and supported across many shell environments.

**Table 3-9. Standard variables**



Key Points: Terminal Types

1. **TERM Variable**:
   * TERM is crucial for screen-based programs like vi, emacs, and more.
2. **Purpose**:
   * It tells applications how to control your terminal’s screen (e.g., cursor movement, scrolling, colors).
3. **Correct Setting**:
   * TERM must match a valid terminal type found in the **terminfo** database (/usr/lib/terminfo).
4. **File Format**:
   * terminfo entries are binary files named after terminal types, organized into subdirectories by the first letter.
5. **Check TERM Value**:
   * Use echo $TERM to see the current setting  
     cd /usr/lib/terminfo
   * ls [first-letter]/[name]\*
6. **Fallback Option**:
   * If no exact match exists, use terminal emulation (e.g., emulate a VT100 or xterm).
7. **Multiple Entries**:
   * Some terminals have multiple entries for submodels/modes—try each and see which works best.
8. **Graphical Terminals**:
   * GUI terminal windows (e.g., **xterm** in X Window System) often auto-set TERM correctly.
9. **Configuration Tip**:
10. Set the appropriate TERM in .bash\_profile like:  
    TERM=vt100

* export TERM

Key Points: The Environment File

1. **Purpose of Environment File (.bashrc)**:
   * Used to define shell variables, options, and aliases **for use in subprocesses**.
   * Ensures that these settings are consistently available across interactive shells.
2. **Relationship with .bash\_profile**:

If settings are moved from .bash\_profile to .bashrc, add the line:  
source .bashrc

* + to .bash\_profile so login shells inherit them.

1. **Naming Convention**:
   * .bashrc is named after the C shell's .cshrc.
   * The suffix rc stands for **"run commands"**—commonly used in UNIX config files.
2. **Best Practices**:
   * Keep **.bash\_profile minimal**—only include:
     + Environment variable exports
     + Commands that **run or produce output** during login
   * Place **aliases, functions, and shell options** in .bashrc.

**Example of a Minimal .bash\_profile**:  
stty stop ^S intr ^C erase ^?

Date

source .bashrc

1. Advanced Notes:
   * Use set -a or set -o allexport to auto-export all assigned variables.
   * The set -k option allows placing variable definitions anywhere in the command line.
2. Modern Systems Note:
   * Most UNIX systems now use a database-based terminal info system rather than flat files
   * Key Points: Customization Hints
3. Try Before You Commit:
   * Test any customization (like variables, aliases, prompts) directly in the shell before adding it to .bash\_profile.
4. Make It Permanent:
   * If a customization works well, append it to .bash\_profile to make it persistent across sessions.
5. Using Shell History to Recall Commands:
   * In emacs-mode: Use CTRL-P or CTRL-R.
   * In vi-mode: Use j, -, or ? to recall previous commands.
6. Append Using echo:

Use echo and redirection to add a command to .bash\_profile without opening a text editor:  
echo 'PS1="\u \!--> "' >> ~/.bash\_profile

1. Use Single Quotes in echo:
   * Prevents the shell from interpreting special characters (e.g., $, ", !).
2. Important Redirection Note:
   * Use >> to append.
   * Avoid >, which would overwrite .bash\_profile.