Bash Basics 5: Redirections & Locale

These slides introduce the basics of working with **Bash**:

1. Intro & Commands
2. Expansions & Metachars
3. Variables & Quoting
4. Interactive Use
5. **Redirections & Locale**
   - **input and output redirection**
   - **locale and collation (sorting) order**

Redirections

Prior to actual execution of an interpreted command line, input and output to/from the commands may be **redirected**.

For example, instead of having output go to the terminal, it might be redirected to a file (so it can be saved).

Redirections are done in the shell, so commands/programs are usually unaware of it: a command writes to standard output as normal, but the output ends up in a file.

We have two kinds of redirections:

- **output redirection** – affects output/writing
- **input redirection** – affects input/reading

Redirections (contd.)

Output redirection:

- syntax: *command* `>FILE`
- syntax: *command* `>>FILE` (for appending)
- causes *command* output (to **standard output**) to be written/appended to `FILE`

Input redirection:

- syntax: *command* `<FILE`
- causes *command* input (from **standard input**) to be read from `FILE`

Redirections (contd.)

Examples:

- `grep printf lab2.c > lab2-printfs`
- `grep printf *.c >> labs-printfs`
- `ls -l *.c | grep Jun | wc -l > count`
- `grep printf < lab2.c`
  (same effect as: `grep printf lab2.c`)
- `tr -d'\r' < dosfile > unixfile`
  (`tr` only reads from stdin and writes to stdout,
  so must use redirection to apply to files)

Redirections (contd.)

Can redirect **standard error** output as well:

- redirect standard error output:
  command `2> FILE`
- redirect standard output and standard error:
  command `&> FILE`
  (this is shorthand for: *command* `>FILE 2>&1`)

Redirections are generally applied to standard input, standard output, and/or standard error.

They can be applied to arbitrary **file descriptors** however:

- syntax: *command* `[N]>FILE`
- syntax: *command* `[N]<FILE`
- `[N]` indicates optional FD (integer)

Locale and Collation Order

Internationalization (**i18n**) affects some aspects of filename expansion and other shell elements.

Internationalization settings are controlled by **locale** settings, which are maintained as *environment variables*, most having the prefix "`LC_`".

The `locale` command will print a list of the locale environment variables and their values.

An important variable for Bash is `LC_COLLATE`, which determines the **collation (sort) order** for characters.

Locale and Collation Order (contd.)

Collation order affects the meaning of character ranges in the `[...]` filename expansion.

For example, we expect `[A-Z]` to match any uppercase letter, since the ASCII collation order is abc...xyzABC...XYZ.

However, with some newer distros, the default collation order can be aAbBcC...xXyYzZ.

This causes `[A-Z]` to be equivalent to `[AbBcC...xXyYzZ]`, i.e., to match all letters but "a".

This could be disastrous if a user executes a command such as "`rm [A-Z]*.data`" as files may unexpectedly be deleted.

Locale and Collation Order (contd.)

Collation order may be set to standard ASCII order by running the following command:

```
export LC_COLLATE=C
```

A collate setting that will cause order issues is:

```
LC_COLLATE=en_US.UTF-8
```

The locale settings can affect other things, such as the *date format* used by `ls` or other programs.