BASH/SHELL Workshop for Data Science

Session II

Making Tea with Makefiles

Agenda

- An example with shell scripts, and a workflow pipeline helped by ‘makefiles’.
- How to copy files using cp command.
- How to use remove command
- How to unpack using tar
- Logical operators, &&, ||

Download Making tea tutorial

makefiles_tea_tutorial.tar

Getting started

1. download the makefiles_tea_tutorial.tar
    1. link → here
2. unpack / extract it either using GUI or via command line, you can do
    1. $ tar xf makefiles_tea_tutorial.tar #tar xf unpacks the folder
    2. cd makefiles_tea_tutorial/
    3. NOTE: on WSL, the files would be download on the Windows side, so from Ubuntu side, you go to /mnt/c/Users/<yourUserName>/Downloads/ and copy files from here
        1. cd /mnt/c/Users/sk/Downloads/
        2. cp makefiles_tea_tutorial.tar ~/   #tilda
        3. cd ~
        4. tar xf makefiles_tea_tutorial.tar
        5. Or, if you choose to unpack it while it is in Downlaods folder from windows (GUI)
        6. then you now copy the unpacked folder like so:
            1. cp -r makefiles_tea_tutorial/ ~/
            2. cd ~

Some tips: create symbolic link to downloads folder

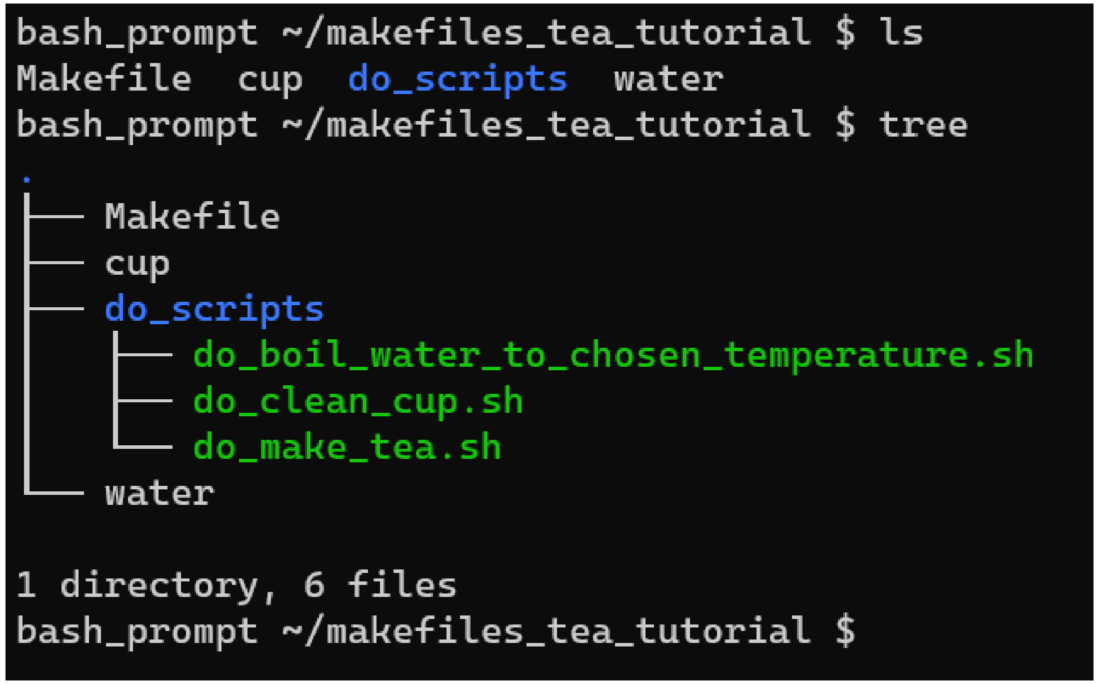
1. TIPs: make a symbolic link from Windows side Downloads and Documents directories to your WSL $HOME folder

1. ln -sf /mnt/c/Users/<yourusername>/Downloads $HOME/wDownloads
2. ln -sf /mnt/c/Users/<yourusername>/Documents $HOME/wDocuments
    1. now you can just cd ~/wDocuments/ and be able to access your windows documents folder.
3. If you are not sure about path to your current directory and want to check it in windows explorer instead ! –
   Then open Windows explorer (GUI) from wherever you are in WSL, do,
   " explorer.exe . " (that is a dot, the current directory!).

Tree command

1. **Tree** command you see above, btw, is a fancy but nice way to look at your directory structure, compared to just doing "ls", you'll have to install it separately in Ubuntu though (or in WSL) like so: "sudo apt install tree")

- Here **is the initial state of the files** (remember, 'cat filename' to quickly see what's in small text-files)

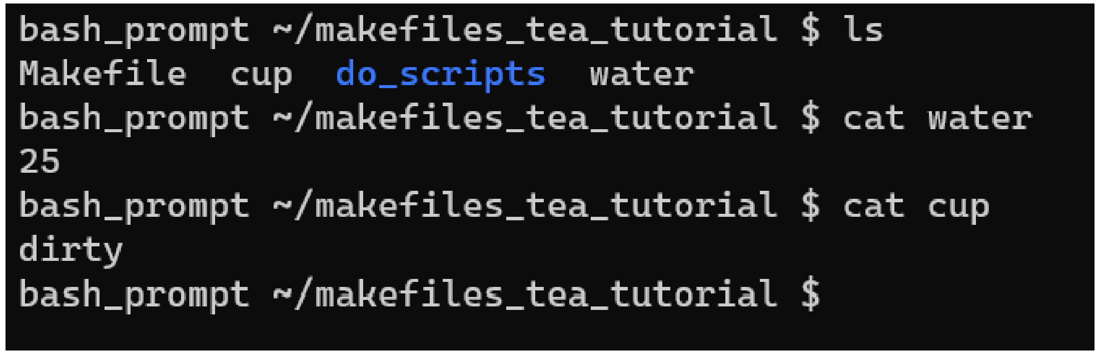


```
bash_prompt ~/makefiles_tea_tutorial $ ls
Makefile  cup  do_scripts  water
bash_prompt ~/makefiles_tea_tutorial $ tree
.
├── Makefile
├── cup
├── do_scripts
│   ├── do_boil_water_to_chosen_temperature.sh
│   ├── do_clean_cup.sh
│   └── do_make_tea.sh
└── water

1 directory, 6 files
bash_prompt ~/makefiles_tea_tutorial $
```

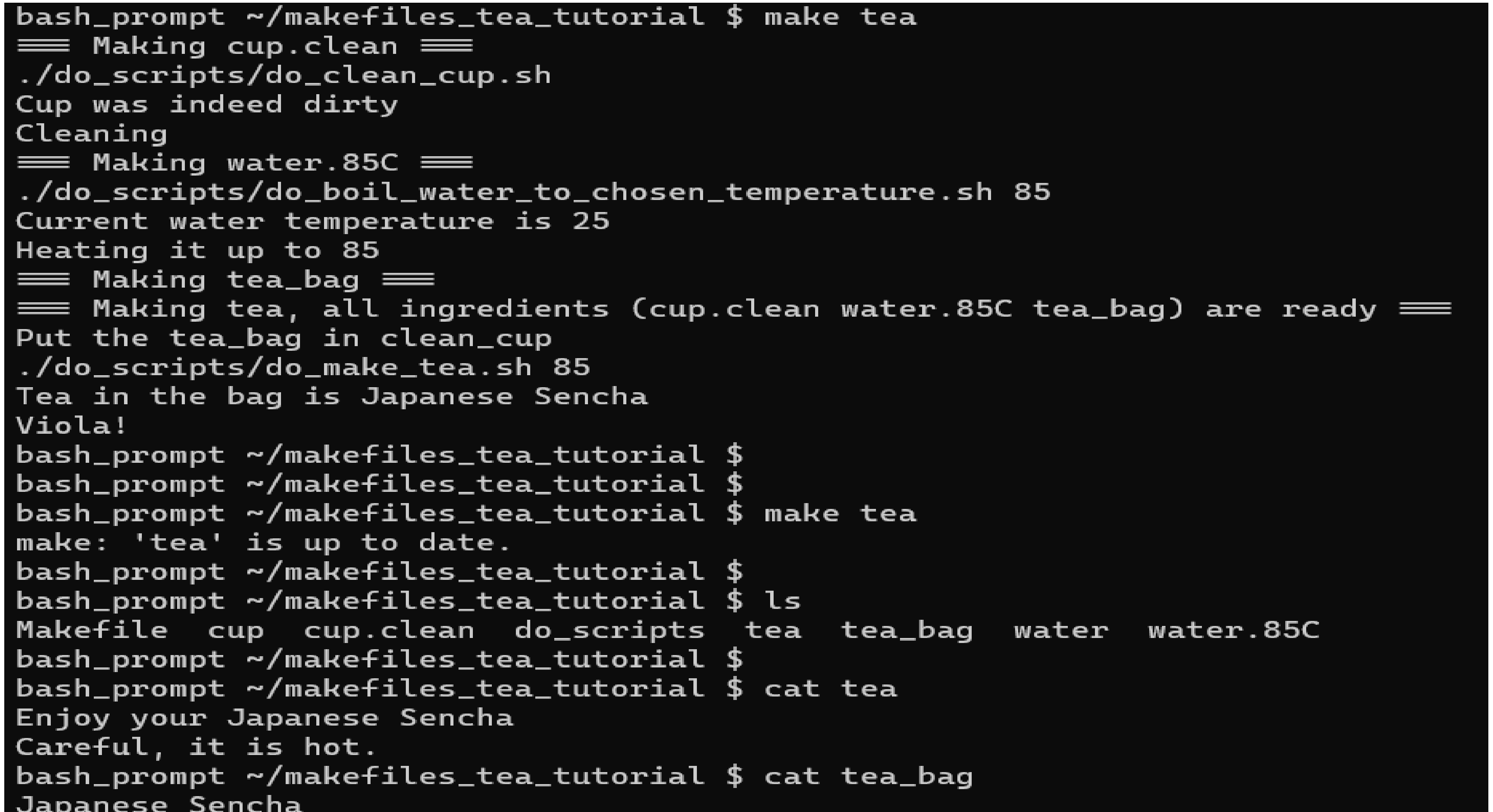
Next… let’s do ‘make tea’

- What do you need for making a tea?
- A water?
- A clean cup?
- A tea bag?
- Lets try the recipe by running the make tea recipe.



```
bash_prompt ~/makefiles_tea_tutorial $ ls
Makefile  cup  do_scripts  water
bash_prompt ~/makefiles_tea_tutorial $ cat water
25
bash_prompt ~/makefiles_tea_tutorial $ cat cup
dirty
bash_prompt ~/makefiles_tea_tutorial $
```

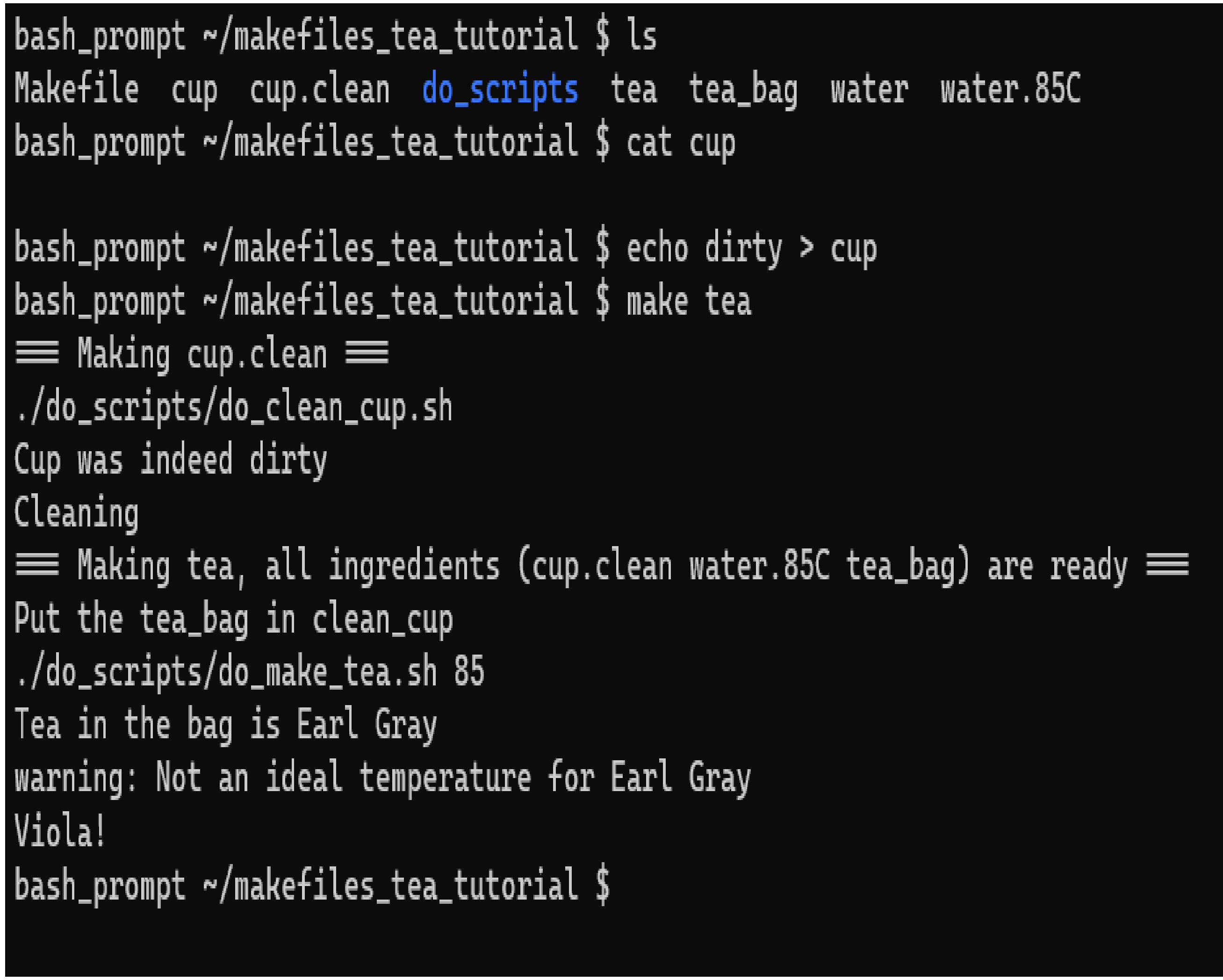
Making the tea..



```
bash_prompt ~/makefiles_tea_tutorial $ make tea
=== Making cup.clean ===
./do_scripts/do_clean_cup.sh
Cup was indeed dirty
Cleaning
=== Making water.85C ===
./do_scripts/do_boil_water_to_chosen_temperature.sh 85
Current water temperature is 25
Heating it up to 85
=== Making tea_bag ===
=== Making tea, all ingredients (cup.clean water.85C tea_bag) are ready ===
Put the tea_bag in clean_cup
./do_scripts/do_make_tea.sh 85
Tea in the bag is Japanese Sencha
Viola!
bash_prompt ~/makefiles_tea_tutorial $
bash_prompt ~/makefiles_tea_tutorial $
bash_prompt ~/makefiles_tea_tutorial $ make tea
make: 'tea' is up to date.
bash_prompt ~/makefiles_tea_tutorial $
bash_prompt ~/makefiles_tea_tutorial $ ls
Makefile  cup  cup.clean  do_scripts  tea  tea_bag  water  water.85C
bash_prompt ~/makefiles_tea_tutorial $
bash_prompt ~/makefiles_tea_tutorial $ cat tea
Enjoy your Japanese Sencha
Careful, it is hot.
bash_prompt ~/makefiles_tea_tutorial $ cat tea_bag
Japanese Sencha
```

Check what output files are generated: ls command

1. okay, some things happened after ‘make tea’, after that, when you type ‘make tea’ again, you see it says ‘tea’ is up to date, and nothing more to be done. (How does it know? it checks the file ‘tea’ and its timestamp relative to who generated it, more on this later)
2. do an ‘ls’, and you’ll see some new files have appeared since we last ran ‘ls’
3. cat tea, and see what’s inside of it.

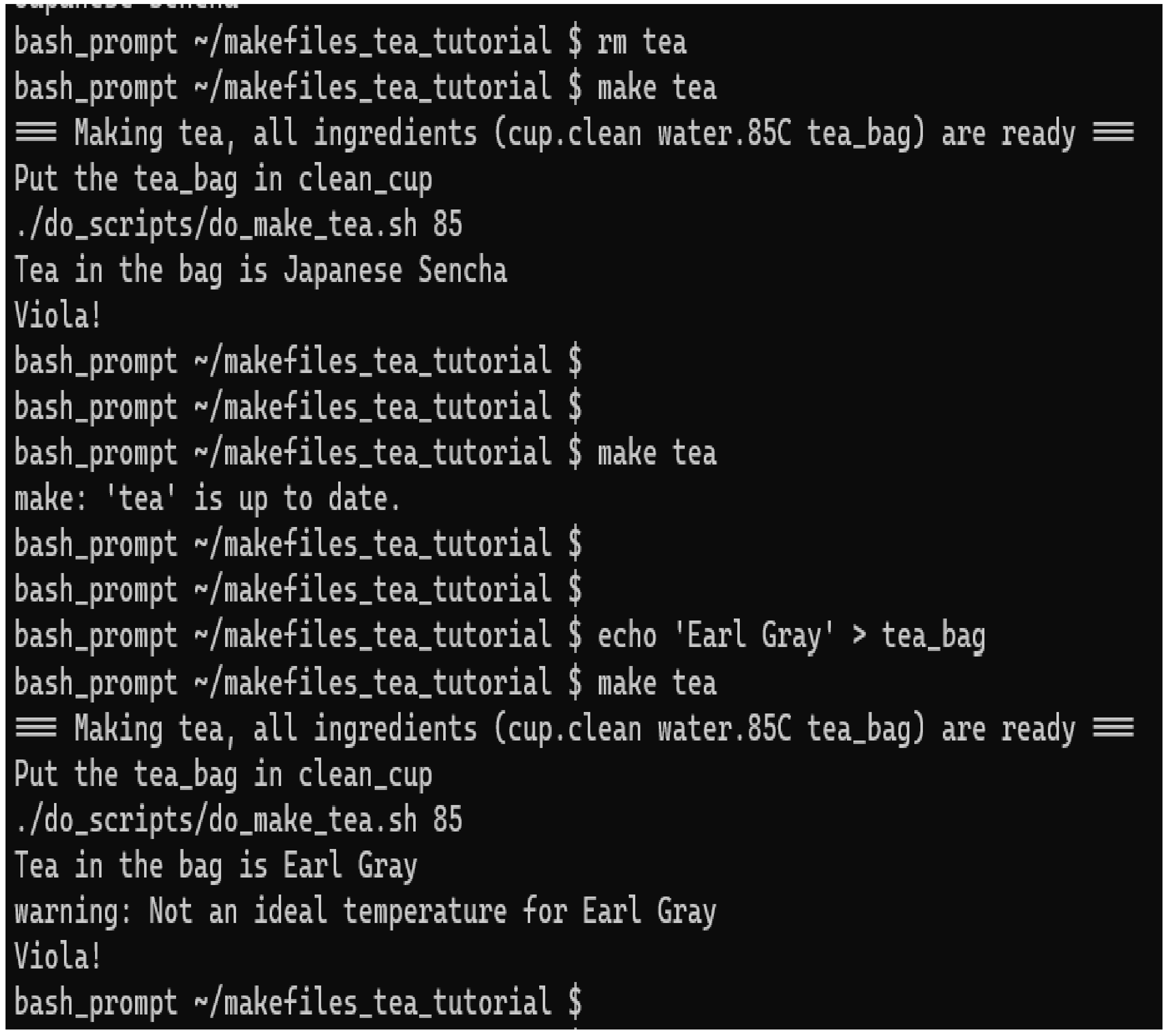


```
bash_prompt ~/makefiles_tea_tutorial $ ls
Makefile  cup  cup.clean  do_scripts  tea  tea_bag  water  water.85C
bash_prompt ~/makefiles_tea_tutorial $ cat cup

bash_prompt ~/makefiles_tea_tutorial $ echo dirty > cup
bash_prompt ~/makefiles_tea_tutorial $ make tea
≡≡ Making cup.clean ≡≡
./do_scripts/do_clean_cup.sh
Cup was indeed dirty
Cleaning
≡≡ Making tea, all ingredients (cup.clean water.85C tea_bag) are ready ≡≡
Put the tea_bag in clean_cup
./do_scripts/do_make_tea.sh 85
Tea in the bag is Earl Gray
warning: Not an ideal temperature for Earl Gray
Viola!
bash_prompt ~/makefiles_tea_tutorial $
```

Deletion

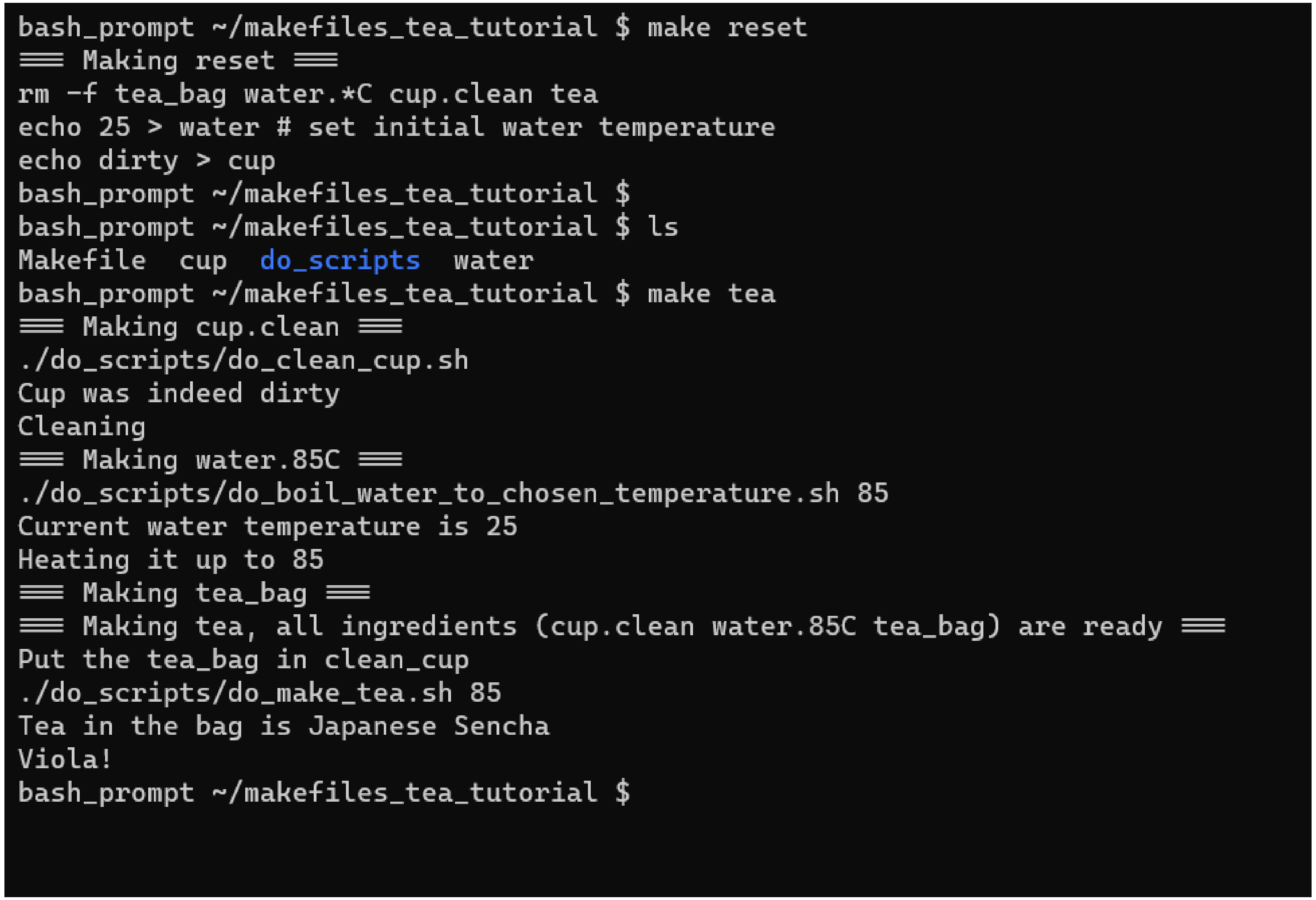
1. let’s delete tea (rm tea) and try make tea again, this time, something happens, but not like the first call to ‘make tea’.... you see only the last stage of making tea happened (since the ingredients in the brackets are ‘already up to date’ or are already ready)
2. If you do ‘make tea’ again, same as before, tea is just made, it won’t be made again, unless we rm tea, OR...something else… let’s change the tea in our tea_bag to “Earl Gray”?, and see what happens. We expect that since the tea was ultimately generated from what’s in the tea bag, if we run this ‘make tea’ , let’s call it, a pipeline, again, it should see the current tea is outdated, and should then make tea with the new tea again. → you see, above, that that indeed happens (you can do ‘cat tea’ to verify).
3. likewise, if the cup were to be made dirty again, we expect since cup.clean is a dependency to make tea (the final target), it should remake tea after cleaning the cup. Let’s see:



```
bash_prompt ~/makefiles_tea_tutorial $ rm tea
bash_prompt ~/makefiles_tea_tutorial $ make tea
=== Making tea, all ingredients (cup.clean water.85C tea_bag) are ready ===
Put the tea_bag in clean_cup
./do_scripts/do_make_tea.sh 85
Tea in the bag is Japanese Sencha
Viola!
bash_prompt ~/makefiles_tea_tutorial $
bash_prompt ~/makefiles_tea_tutorial $
bash_prompt ~/makefiles_tea_tutorial $ make tea
make: 'tea' is up to date.
bash_prompt ~/makefiles_tea_tutorial $
bash_prompt ~/makefiles_tea_tutorial $
bash_prompt ~/makefiles_tea_tutorial $ echo 'Earl Gray' > tea_bag
bash_prompt ~/makefiles_tea_tutorial $ make tea
=== Making tea, all ingredients (cup.clean water.85C tea_bag) are ready ===
Put the tea_bag in clean_cup
./do_scripts/do_make_tea.sh 85
Tea in the bag is Earl Gray
warning: Not an ideal temperature for Earl Gray
Viola!
bash_prompt ~/makefiles_tea_tutorial $
```

Reset

- To reset, and play again, do
- notice the ‘ls’ after ‘make reset’ returns everything to ‘factory state’
- Now, let’s take a look inside the makefile.



```
bash_prompt ~/makefiles_tea_tutorial $ make reset
=== Making reset ===
rm -f tea_bag water.*C cup.clean tea
echo 25 > water # set initial water temperature
echo dirty > cup
bash_prompt ~/makefiles_tea_tutorial $
bash_prompt ~/makefiles_tea_tutorial $ ls
Makefile  cup  do_scripts  water
bash_prompt ~/makefiles_tea_tutorial $ make tea
=== Making cup.clean ===
./do_scripts/do_clean_cup.sh
Cup was indeed dirty
Cleaning
=== Making water.85C ===
./do_scripts/do_boil_water_to_chosen_temperature.sh 85
Current water temperature is 25
Heating it up to 85
=== Making tea_bag ===
=== Making tea, all ingredients (cup.clean water.85C tea_bag) are ready ===
Put the tea_bag in clean_cup
./do_scripts/do_make_tea.sh 85
Tea in the bag is Japanese Sencha
Viola!
bash_prompt ~/makefiles_tea_tutorial $
```

Appendix: Notes on some commands from the tutorial

- tar command –
- The standard UNIX archiving utility. Some useful tar options:
- 1. −c create (a new archive)
- 2. −−delete delete (files from the archive)
- 3. −r append (files to the archive)
- 4. −t list (archive contents)
- 5. −u update archive
- 6. −x extract (files from the archive)
- 7. −z gzip the archive

make Command – makefile

- # A makefile is a list of recipies to make various targets. It is a GNU make utility to maintain groups of programs.
- # That's about it.
- # Syntax is as follows:#
- # target: dependency1 dependency2 ....#
- <TAB>command1#
- <TAB>command2#
- <TAB>...#
- # Note that each dependency_i on the right, is a target defined elsewhere (below / above, order doesn't matter)# A recipe for a target is executed completely / correctly, if all the commands succeed (ie., with exit codes as SUCCESS, which is integer value 0)#
- # Commands can execute R code or python code , etc. #
- SYNOPSIS
-        make [OPTION]... [TARGET]...

Thanks for attending session II !