Part II

Shell Config

Special Directories

- `.` current directory
- `..` parent directory
- `~` own home directory
- `~user` home directory of user
- `-` previous directory

```
cd /usr/local/bin
pwd
cd ~
pwd
cd -
pwd
cd ..
pwd
```

Aliases and Variables

alias - short form for any command

```
alias
alias ll='ls -l'
```

environment variables

- Variables are not only within scripts, but also in the shell
- By setting a variable, it is present in the current script/shell
- By exporting it (`export VARIABLE=value`), it is also present at all child processes (NOT at the parent)
- Many environment variables are already set by default

Aliases and Variables

alias - short form for any command

```
alias
alias ll='ls -l'
```

environment variables

- Variables are not only within scripts, but also in the shell
- By setting a variable, it is present in the current script/shell
- By exporting it (`export VARIABLE=value`), it is also present at all child processes (NOT at the parent)
- Many environment variables are already set by default

```
PATH=/home/neckel/bin/:$PATH; echo $PATH
env
```

local variables

- `set` shows all local and global variables and functions
- `unset` deletes a variable

Bash config

profile and rc

- especially aliases are used in every session
  $\Rightarrow$ should not be defined each time
- You might even run arbitrary code on login or logout
  (e.g., clean up (kill jobs) on logout)
- `profile` is executed for login shells (everything with password or keys)
- `bashrc` is executed for interactive non-login shells (e.g., new terminal window)

Bash config

profile and rc

- especially aliases are used in every session
  $\Rightarrow$ should not be defined each time
- You might even run arbitrary code on login or logout
  (e.g., clean up (kill jobs) on logout)
- `profile` is executed for login shells (everything with password or keys)
- `bashrc` is executed for interactive non-login shells (e.g., new terminal window)

Files

- `/etc/profile`
- `/etc/bash.bashrc`
- `$HOME/.bash_profile`
- `$HOME/.bashrc`
- `$HOME/.bash_logout`

Pipes

- stdin/stdout: Standard input/output for programs
- Most Linux programs read from stdin and write to stdout
- Pipes are used to redirect input and output
- `cmd1 | cmd2` connect the output of cmd1 with the input of cmd2
- `cmd > file` redirect the output of cmd to file

Pipes

- stdin/stdout: Standard input/output for programs
- Most Linux programs read from stdin and write to stdout
- Pipes are used to redirect input and output
- `cmd1 | cmd2` connect the output of cmd1 with the input of cmd2
- `cmd > file` redirect the output of cmd to file
- `cmd 2> file` redirect stderr of cmd to file
- `cmd > file 2>&1` redirect stdout to file and stderr to stdout
- `cmd &> file` redirect all output to file
- `cmd >> file` redirect the output of cmd and append it to file
- `cmd < file` use the content of file as stdin for cmd
- `cmd <<Endmark` Read from stdin until Endmark is inserted in a seperate line
- `;` command separator

Wildcards

- Wildcards are expanded by the shell
- `*` – zero or more characters
- `?` – exactly one character
- `[abcd]` – one of the characters a-d
- `[a-d]` – same
- `[!a-d]` – any other character
- `{first,second}` – either `first` or `second`
- Similar patterns exist in other contexts as well (compare regular expressions), but always a bit different. . . :-(

Part III

Bash Advanced: Regular Expressions and More

Regular Expressions???

Regular Expressions???

The set of regular languages over an alphabet Σ and the corresponding regular expressions are defined recursively as follows:

- The empty language Ø is a regular language, and the corresponding regular expression is Ø.
- The empty string $\{\wedge\}$ is a regular language, and the corresponding regular expression is $\wedge$.
- For each a in Σ, the singleton language $\{$ a $\}$ is a regular language, and the corresponding regular expression is a.
- If A and B are regular languages, and r1 and r2 are the corresponding regular expressions,
  - Then A U B (union) is a regular language, and the corresponding regular expression is (r1+r2)
  - AB (concatenation) is a regular language, and the corresponding regular expression is (r1r2)
  - A* (Kleene star) is a regular language, and the corresponding regular expression is (r1*)

Regular Expressions!!!

What does all this mean to you, as a user? Absolutely nothing. As a user, you don't care if it's regular, nonregular, unregular, irregular, or incontinent. So long as you know what you can expect from it, you know all you need to care about.

Jeffrey Friedl, author of *Mastering Regular Expressions*

Regular Expressions!!!

What does all this mean to you, as a user? Absolutely nothing. As a user, you don't care if it's regular, nonregular, unregular, irregular, or incontinent. So long as you know what you can expect from it, you know all you need to care about.

Jeffrey Friedl, author of *Mastering Regular Expressions*

Did you ever...

- ...search for a character or string in a text file?
- ...use tab for auto-completion?
- ...use the * in a terminal for selecting a group of files?
- ...

Then you've already somehow used regular expressions.

Regular Expressions

- When searching for a string, exactly the given character sequence is searched
- Regular expressions: a powerful tool to find patterns
- Additionally to ordinary characters, which stand for themselves special characters are used which are interpreted in a special way.
- The exact syntax for special characters differs between different implementations

Regular Expressions

- When searching for a string, exactly the given character sequence is searched
- Regular expressions: a powerful tool to find patterns
- Additionally to ordinary characters, which stand for themselves special characters are used which are interpreted in a special way.
- The exact syntax for special characters differs between different implementations
- Example to identify an email address:

  ```
  [^@]\+@.\+\.[^.]\+
  ```

- regular expressions usually look very cryptic!
- But imagine you would have to write a normal program to identify an email address!
- Regular expressions can be used to find/replace groups of strings which can be described by a pattern

Special Characters for Regular Expressions

- *char* – A character maching itself
- * matches zero or more occurences (greedy!) of the previous expression
- \+ matches one or more occurences (GNU extension)
- \? matches zero or one occurence (GNU extension)
- \{i\} matches i occurences
- \{i,\} matches i or more occurences
- \{i,j\} matches i to j occurences
- . matches an arbitrary character

Special Characters for Regular Expressions

- *char* – A character maching itself
- * matches zero or more occurences (greedy!) of the previous expression
- \+ matches one or more occurences (GNU extension)
- \? matches zero or one occurence (GNU extension)
- \{i\} matches i occurences
- \{i,\} matches i or more occurences
- \{i,j\} matches i to j occurences
- . matches an arbitrary character
- ^ matches the beginning of a string
- $ matches the end of a string
- [list] matches a single character from the list ([^list] any character not in the list)
- \n matches newline
- \(\) defines a group, reuse via \1 (1st group), \2 (2nd group)
- \| allows for a logical OR

Character Classes

- [:digit:] 0 to 9 (alternative: [0-9]).
- [:alnum:] alphanumeric character 0-9 or A-Z or a-z.
- [:alpha:] character A-Z or a-z.
- [:xdigit:] Hexadecimal notation 0-9, A-F, a-f.
- [:punct:] Punctuation symbols, e.g. . , ” ’ ? ! ; : # $ % & ( )
- [:print:] Any printable character.
- [:space:] whitespace (space, tab, ...).
- [:upper:] uppercase character A-Z (alternative: [A-Z]).
- [:lower:] lowercase character a-z (alternative: [a-z]).

Character Classes

- [:digit:] 0 to 9 (alternative: [0-9]).
- [:alnum:] alphanumeric character 0-9 or A-Z or a-z.
- [:alpha:] character A-Z or a-z.
- [:xdigit:] Hexadecimal notation 0-9, A-F, a-f.
- [:punct:] Punctuation symbols, e.g. . , ” ’ ? ! ; : # $ % & ( )
- [:print:] Any printable character.
- [:space:] whitespace (space, tab, ...).
- [:upper:] uppercase character A-Z (alternative: [A-Z]).
- [:lower:] lowercase character a-z (alternative: [a-z]).

Attention: Additional brackets are necesssary! $[[:digit:]] \equiv [0-9]$

programs using regular expressions

- `grep` print lines matching a pattern
- `grep -i` case insensitive
- `grep -v` invert match
- `grep -n` additionally print line number
- `grep -r` work recursively
- `tr` translate: `echo "lower/UPPER"| tr "A-Z""a-z"`
- `sed` stream editor
- `awk` pattern scanning and processing language
- `find` search for files in a directory hierarchy
- ...
- most editors can handle regular expressions

Operating on Files

less is more than more

- `more` displays the content of text files pagewise
- only downward-scrolling is possible
- `less` is an extended and more flexible `more`

Operating on Files

less is more than more

- `more` displays the content of text files pagewise
- only downward-scrolling is possible
- `less` is an extended and more flexible `more`

outputting files

- `cat` outputs the content of a file to stdout (try `tac`)
- `cat` concatenates several files
- `head` outputs the first part of files
- `tail` outputs the last part of files

```
for i in `seq 30`; do echo $i >> temp.txt; sleep 1; \
done &
tail -f temp.txt
```

Operating on Files (2)

- `file` find out more about file type
- `cmp` Compare files byte by byte
- `diff` Compare files line by line (graphical: kdiff3)
- `tar` pack and compress files (try `tar -xzf` and `tar -czf`)
- `sort` sort lines of text files (and write to stdout)
- `wc` print newline, word, and byte counts for each file
- `cut` print selected parts of lines from each file

```
echo "1;2;3;4" | cut -b 4-5
echo "1;2;3;4" | cut -d ";" -f 3
cut -d ' ' -f 1 cut.txt
```

find

- Tool to retrieve recursively(!) files and directories depending on search criteria
- Perform operations on the files found
- In connection with other bash tools perform tasks such as
  - Find all files not older than two days
  - Find all .txt files containing the phrase ”Hello World!”
  - Find all .bak files and remove them

find

- Tool to retrieve recursively(!) files and directories depending on search criteria
- Perform operations on the files found
- In connection with other bash tools perform tasks such as
  - Find all files not older than two days
  - Find all .txt files containing the phrase ”Hello World!”
  - Find all .bak files and remove them
- General syntax:

  `find [path...] [expression]`

  An expression can be
  - Options, such as `--maxdepth n`
  - Tests, such as `-amin n` (last access less than n minutes ago)
  - Actions, such as `-delete`

  If path is not specified, assume `./`

find (2)

```
wget http://www5.in.tum.de/lehre/vorlesungen/\
progcourse/bashpython/ss19/Bash/story.tgz
tar -xzf story.tar.gz
```

Examples

```
#!/bin/bash
find -maxdepth 2 -name "*.txt"
find ./ -amin -500 -iname "*.TXT"
find -atime +1 -name "*.txt"
find -path "*folder1*"
find -size +400c -name "file*.*" -ls
find -name "*.txt" -printf "%h, %f\n"
find -name "file??.txt" -exec grep -H "Bilbo" {} \;
find -regex '.*[^0-9][0-4].txt'
find -name "*.txt" -delete  # take care!!
```

Executing Commands

- For executing a single command, it first has to be found!
- Either path is provided, or command is in $PATH
- `which`, `whereis` and `type`: find out about program type and path
- For commands with more than one line:

```
echo "This␣command␣prints␣a␣lot␣of␣useless␣text␣\
which␣does␣not␣fit␣into␣one␣line"
```

Executing Commands

- For executing a single command, it first has to be found!
- Either path is provided, or command is in $PATH
- `which`, `whereis` and `type`: find out about program type and path
- For commands with more than one line:

```
echo "This␣command␣prints␣a␣lot␣of␣useless␣text␣\
which␣does␣not␣fit␣into␣one␣line"
```

multiple commands

- One way to execute several commands: connecting IO via pipes
- `cmd1; cmd2; ...` executing several commands
- `cmd1 && cmd2 && ...` execute following commands if previous successful
- `cmd1 || cmd2 || ...` execute following commands if previous fails

Part IV

Working Remotely And More

Working remotely - ssh, scp, screen and unison

ping

- `ping hostname`
- Useful to find out whether some machine is down or has no connection.

Working remotely - ssh, scp, screen and unison

ping

- `ping hostname`
- Useful to find out whether some machine is down or has no connection.

ssh and scp - security by encryption

- Developed 1995 to allow secure connections between computers
- `ssh username@computername` log in onto computername as username
- `scp -r * username@computername:<path_rel_to_home>`
- `scp *.txt username@computername:/<absolute_path>`
- `scp "username@computername:path/*.*" ./` – copy all remote files in folder `path` to current directory
  - Quotes necessary for remote wildcards

Synchronising data – Unison

- very nice to sync 2 locations
- GUI and backup support
- 2-way sync (including merge etc.)
- supports Windows and Linux
- needs gtk and ssh
- no active development any longer
- download and infos:
  `http://www.cis.upenn.edu/~bcpierce/unison/`

Management of processes

- `nice -n <value> <command>` starts command with lower priority
- `renice <value> -p <processID>` sets priority to new value
- `top` shows processes with top computing time
- within `top`, `r` can renice a process
- `nohup` run a command immune to hangups
- `ps -ef` list all processes
- `pidof` find out the process id for a program name
- `kill <ID>` kills process with the given Id (-9 to force)
- `killall <name>` kills processes with given name
- `bg` and `fg` send programms to background/foreground

Working remotely - screen

- during a typical session, many terminals are used
- if working remotely $\Rightarrow$ opening many ssh connections
- for each login/logout, everything has to be closed/opened again
- `screen` solves all these problems by creating virtual terminals

Working remotely - screen

- during a typical session, many terminals are used
- if working remotely $\Rightarrow$ opening many ssh connections
- for each login/logout, everything has to be closed/opened again
- `screen` solves all these problems by creating virtual terminals
  - `screen` creates a new terminal
  - `Ctrl-a d` or `screen -d` detach the screen
  - `Ctrl-a D D` detach the screen and log out
  - `screen -r` reattach the screen
  - `Ctrl-a A` give a name to a terminal
  - `screen -t <title>` creates a new terminal with specified title
  - `Ctrl-a w` show the currently opened terminals
  - `Ctrl-a k` kill a terminal

Some more commands you might want to know

```
basename "this/is/the/path/to/a.file"
dirname "this/is/the/path/to/a.file"
time sleep 3
time for i in `seq 100000`; do a=$[3*$i]; done
users
groups
du -sh
df -h
ln -s <file> <link>
```