Command line

Like a pro

Why bother?

“Give me six hours to chop down a tree and I will spend the first four sharpening the axe”

- Lincoln

> You use the command line **every day**

> **powerful** and **flexible**

> Your knowledge will never go stale

> It’s easy to not bother

Sections

- Time savers
- Multi-part commands
- Handling text files
- File system
- Jq
- CLIs for BW tools
- Aliases
- Scripts
- Other

Time savers

Stop typing things!

`tab` for autocomplete

`control-r` to reverse search

Don't remember a command on a shared server? CTRL-R to see what other people have typed!

`alias` to create shortcuts for commands

Use them to

`scripts` to automate common jobs

More on this later

`!!` repeats the last command

Navigating input line

`control-left/right` move by word, instead of by character

`control-a/e` or **`home/end`** to move to the beginning or end of the line

`control-u` delete up to the cursor

`control-k` deletes everything after the cursor

Multi-part commands

Chaining commands

`&&` chains commands together (only if the first command succeeds)

`||` only if the last command fails

`&` execute both commands simultaneously

`\` split a command over multiple lines for readability

Direct your output somewhere

`>` into a file

`>>` append to a file

`|` into the next command

`> /dev/null` to discard output

Redirecting: stderr v stdout

The “error” and “output” of a command can be handled different.

```
Grep foo my-file.txt > result.txt
```
The result of the grep is saved to result.txt
If the grep errors (e.g. the file can’t be found), the error is printed to the terminal

```
Grep foo my-file.txt 2> errors.txt
```
The errors are saved to file, but the result of the grep is printed

```
Grep foo my-file.txt &> all.txt
```
Both errors are results are saved to the same file

```
Grep foo my-file.txt > result.txt 2> errors.txt
```
Errors and results are saved to different files

<(help I don’t understand this one)

< to redirect as input to another command. Similar to pipe, but can use multiple commands as inputs

Handling text files

Reading files

`vim` to edit

`less` to view

CTRL-F to watch the end of the file

`zless` to view zipped files

`cat` prints the file contents to the terminal

`grep` to search through files

Zgrep to for zipped files

`head/tail` to print the start or end of a file

Manipulating files

`wc` to count words

Wc -l to count lines

`sort` to sort the output

-n to sort numerically

`uniq` to get unique lines

Requires sorted data

Comparing files

Grep

```
-i
-o
-v
-r
-c
-C 5
-A 5
-B 5
```

File System

Get more out of your most used commands

Cd

```
cd -
Cd ..
cd ~/foo/bar
cd
```

Ls

```
Ls -a
Ls -l
Ls -lh
Ls -I
Ls -R
Ls -t
Ls -altr
```

jq

Handle json

Getting fields

`jq .`	get all fields (and format it nicely)
jq .url	get a field by name
jq .foo.bar	gets nested field by name
jq '.url, .text'	gets multiple fields
jq ..	gets every value, flattened
jq ..|.name?	Gets nested values without specifying the full path

Other things

`jq -S .` sort keys alphabetically

`|` feeds one filter from one into another
`.a|.b` is equivalent to `.a.b`

Addition (+) and subtraction (-) to manipulate numbers, concatenate arrays and strings. Also multiplication, division, and modulo

Other things

jq ‘keys’ lists top level keys

jq ‘.siteMetrics | keys’ lists nested keys

jq '. | select(.resourceType=="premium_reddit") | .parsedText'
get the text for every reddit element

arrays

jq .[1]	gets the first element in an array
jq .[-1]	gets the last element in the array
jq .[0:5]	gets the first 5 elements
jq .[]	gets the whole array
jq .[].foo	gets all the foos from the array
jq '.foos | length'	gets the length of the array

Creating json

jq ‘{name: .authorMetrics.name, site: .siteMetrics.domain}’

create a new json with the fields name and site, based on the values of the old json

jq {foo, bar} is equivalent to jq {foo: .foo, bar: .bar}

CLIs for our tools

Kafka CLI

Read from kafka: ~/opt/kafka-1/bin/kafka-avro-console-consumer --topic mention.operations.shard.1 --bootstrap-server localhost:9092

--from-beginning
--max-messages
--offset

Get consumer groups: ~/opt/kafka-1/bin/kafka-consumer-groups --bootstrap-server localhost:9092 --list

Get lag/offset: ~/opt/kafka-1/bin/kafka-consumer-groups --bootstrap-server localhost:9092 --describe --group resource-metric-storage-consumer

Hbase

lrdb-reader GET http://foo.com/123

There’s also a full hbase shell: **hadoopctl hbase shell**

(I wouldn’t recommend it unless you have to, though)

Solr

Search everything in a mention cluster from the command line!

https://github.com/BrandwatchLtd/solr-cli

E.g. solr-mentions docs dev_mentions0 --query-id 19

AWS cli

- S3 CLI uses ls/mv/cp/rm to makeit feel familiar
- Other aws services use the same CLI
  https://docs.aws.amazon.com/cli/latest/reference/

Things I didn’t even know before now

Xargs

Commands in brackets are executed in subshells. This is useful to do things like (cd /some/other/dir && other-command) without changing directory

Aliases

Making aliases

`alias mci=’mvn clean install”` creates a temporary alias

Add to .bashrc, .bash_aliases or .bash_profile to make it permanent

`alias` lists all your aliases and their commands

`alias foo` shows you the command for the foo alias

I can never be bother to make them

In bashrc

```
als() {
    echo "alias ${1}='${2}'" >> $HOME/.bash_aliases
}
```

Then to store a new alias any time

```
als mynewalias “echo ‘hello world’”
```

Or simply assign your last command to an permanent alias

```
als myalias "!!"
```

Aliases for shell customization

```
ls='ls --color=auto'

ansible-playbook='cd ~/git/ansible-app && ansible-playbook'

ansible-local="cd ~/git/ansible-app && ansible-playbook -e 'branch=LOCAL' "
```

Aliases to make new commands

```
tokenize='tr -s " " | cut -d " "'

gitlog='git log --oneline | head -20'
```

Aliases for common commands

```
ansible-playbook='cd ~/git/ansible-app && ansible-playbook'

ansible-local="cd ~/git/ansible-app && ansible-playbook -e 'branch=LOCAL' "

mci=’mvn clean install’

mcis=’mvn clean install -DskipTests’
```

Aliases as notes

```
Alias solrStage="ssh stage -NL 12345:rn1204:8180"

Alias recentBackfills="echo "select bf.brand_id, bf.status, br.name from
backfills bf join brands br on br.id=bf.brand_id order by bf.id desc limit
5" | db"

alias lagForTopic='~/opt/kafka-1/bin/kafka-consumer-groups
--bootstrap-server localhost:9092 --describe --group'

alias kafkaConsumerGroups='~/opt/kafka-1/bin/kafka-consumer-groups
--bootstrap-server localhost:9092 --list'

mumsnet-prod-ls='gsutil ls gs://bw-prod-mumsnet-data0'

avroToJson='java -jar ~/avro-tools-1.9.0.jar tojson'
```

Aliases as automation

gitB='git checkout master && git pull && git checkout -b '

gitAmend='git add -p && git commit --amend --reuse-message HEAD && git push upstream -f'

gitRebaseMaster='git checkout master && git pull && git checkout - && git rebase -i master'

alias deploycrawler='cd ~/git/crawler && git pull && mvn clean install -DskipTests && cd ~/git/ansible-app && ansible-playbook app_redeploy_crawler.yml -e '\"branch=LOCAL'\"'

Scripts

Git scripts

gitCheckoutPr 17

```
git fetch origin refs/pull/$1/head:pr$1 && \
git checkout pr$1
```

gitClone mention-service

```
cd ~/git &&
git clone git@github.com:BrandwatchLtd/$1.git
cd $1
git remote add upstream git@github.com:laura-neale/$1.git
```

Other

Bash profile

Formatting the prompt

Path

Aliases

Other

History (and history n) prints out your command history

Add # to the start of your command to enter it as a comment (visible in history)

ssh

Save your commonly used hosts to **~/.ssh/config**

scp (or ftp, or filezilla) to copy files across hosts

ssh -NL 1234:rn1204:8180 bwstage@build-stage to forward ports

Managing your system

Ps (-aux) to list the processes that are running

Jps to list just java processes

Kill (-9) to terminate a process

df/du to see disk usage

Free/Top to see memory usage

Netstat or **telnet** to see what’s using which port