Study Guide: Engineering Productivity Tips with Git, Bash and Vim

Afshine Amidi and Shervine Amidi

August 14, 2020

Working in groups with Git

❑ **Overview** – Git is a version control system (VCS) that tracks changes of different files in a given repository. In particular, it is useful for:

- keeping track of file versions

- working in parallel thanks to the concept of branches

- backing up files to a remote server

❑ **Getting started** – The table below summarizes the commands used to start a new project, depending on whether or not the repository already exists:

Case	Action	Command	Illustration
No existing repository	Initialize repository from local folder	`git init`	
Repository already exists	Copy repository from remote to local	`git clone git_address`	path/to/address.git

❑ **File check-in** – We can track modifications made in the repository, done by either modifying, adding or deleting a file, through the following steps:

Step	Command	Illustration
1. Add modified, new, or deleted file to staging area	`git add file`	
2. Save snapshot along with descriptive message	`git commit -m 'description'`	description

❑ **Sync with remote** – The following commands enable changes to be synchronized between remote and local machines:

Action	Command	Illustration
Fetch most recent changes from remote branch	`git pull name_of_branch`	
Push latest local changes to remote branch	`git push name_of_branch`	

❑ **Parallel workstreams** – In order to make changes that do not interfere with the current branch, we can create another branch name_of_branch as follows:
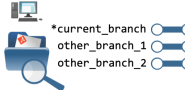
Git

```
git checkout -b name_of_new_branch   # Create and checkout to that branch
```

Depending on whether we want to incorporate or discard the branch, we have the following commands:

Action	Command	Illustration
Merge with initial branch	`git merge initial_branch`	current_branch name_of_branch
Remove branch	`git branch -D name_of_branch`	name_of_branch

❑ **Tracking status** – We can check previous changes made to the repository with the following commands:

Action	Command	Illustration
Check status of modified file(s)	`git status`	Staged changes / Unstaged changes / Untracked files
View last commits	`git log --oneline`	hf06f35 Change logic / 0ey07e0 Rename folder / 18887nd Update file / current_branch
Compare changes made between two commits	`git diff commit_1 commit_2`	commit_1 commit_2 / + added_line / − removed_line
View list of local branches	`git branch`	*current_branch / other_branch_1 / other_branch_2

❏ **Canceling changes** – Canceling changes is done differently depending on the situation that we are in. The table below sums up the most common cases:

Case	Action	Command	Illustration
Unstaged	Revert file to last commit	`git checkout -- file`	file
Staged	Remove file from staging area	`git reset HEAD file`	file
Committed	Go back to a previous commit	`git reset --hard prev_commit`	prev_commit / HEAD HEAD

❏ **Structure of folders** – It is important to keep a consistent and logical structure of the project. One example is as follows:

Structure

```
my_project/
    analysis/
        graph/
        notebook/
    data/
        query/
        raw/
        processed/
    modeling/
```

```
    method/
        tests
    README.md
```

Working with Bash

❏ **Basic terminal commands** – The table below sums up the most useful terminal commands:

Category	Action	Command
Exploration	Display list of files (including hidden ones)	`ls (-a)`
	Show current directory	`pwd`
	Show content of file	`cat path_to_file`
	Show statistics of file (lines/words/characters)	`wc path_to_file`
File management	Make new folder	`mkdir folder_name`
	Change directory to folder	`cd path_to_folder`
	Create new empty file	`touch filename`
	Copy-paste file (folder) from origin to destination	`scp (-R) origin destination`
	Move file/folder from origin to destination	`mv origin destination`
	Remove file (folder)	`rm (-R) path`
Compression	Compress folder into file	`tar -czvf comp_folder.tar.gz folder`
	Uncompress file	`tar -xzvf comp_folder.tar.gz`
Miscellaneous	Display message	`echo "message"`
	Overwrite / append file with output	`output > file.txt / output >> file.txt`
	Execute command with elevated privileges	`sudo command`
	Connect to a remote machine	`ssh remote_machine_address`

❏ **Chaining** – It is a concept that improves readability by chaining operations with the pipe | operator. The most common examples are summed up in the table below:

Action	Command
Count number of files in a folder	`ls path_to_folder	wc -l`
Count number of lines in file	`cat path_to_file	wc -l`
Show last n commands executed	`history	tail -n`

❐ **Advanced search** – The `find` command allows the search of specific files and manipulate them if necessary. The general structure of the command is as follows:

Bash
```
find path_to_folder/. [conditions] [actions]
```

The possible conditions and actions are summarized in the table below:

Category	Action	Command
Conditions	Certain names, regex accepted	`-name 'certain_name'`
	Certain file types (d/f for directory/file)	`-type certain_type`
	Certain file sizes (c/k/M/G for B/kB/MB/GB)	`-size file_size`
	Opposite of a given condition	`-not [condition]`
Actions	Delete selected files	`-delete`
	Print selected files	`-print`

Remark: the flags above can be combined to make a multi-condition search.

❐ **Changing permissions** – The following command enables to change the permissions of a given `file` (or folder):

Bash
```
chmod (-R) three_digits file
```

with `three_digits` being a combination of three digits, where:

- the first digit is about the owner associated to the file
- the second digit is about the group associated to the file
- the third digit is anyone irrespective of their relation to the file

Each digit is one of (0, 4, 5, 6, 7), and has the following meaning:

Representation	Binary	Digit	Explanation
---	000	0	No permission
r--	100	4	Only read permission
r-x	101	5	Both read and execution permissions
rw-	110	6	Both read and write permissions
rwx	111	7	Read, write and execution permissions

For instance, giving read, write, execution permissions to everyone for a `given_file` is done by running the following command:

Bash
```
chmod 777 given_file
```

Remark: in order to change ownership of a file to a given user and group, we use the command `chown user:group file`.

❐ **Terminal shortcuts** – The table below summarizes the main shortcuts when working with the terminal:

Action	Command
Search previous commands	`Ctrl + r`
Go to beginning / end of line	`Ctrl + a` / `Ctrl + e`
Remove everything after the cursor	`Ctrl + k`
Clear line	`Ctrl + u`
Clear terminal window	`Ctrl + l`

Automating tasks

❐ **Create aliases** – Shortcuts can be added to the `~/.bash_profile` file by adding the following code:

Bash
```
shortcut="command"
```

❐ **Bash scripts** – Bash scripts are files whose file name ends with `.sh` and where the file itself is structured as follows:

Bash
```
#!/bin/bash

... [bash script] ...
```

❐ **Crontabs** – By letting the day of the month vary between 1-31 and the day of the week vary between 0-6 (Sunday-Saturday), a crontab is of the following format:

Terminal
```
   *          *            *            *             *
minute     hour          day         month          day
                       of month                   of week
```

❐ **tmux** – Terminal multiplexing, often known as tmux, is a way of running tasks in the background and in parallel. The table below summarizes the main commands:

Category	Action	Command
Session management	Open a new / last existing session	`tmux / tmux attach`
	Leave current session	`tmux detach`
	List all open sessions	`tmux ls`
	Remove `session_name`	`tmux kill-session -t session_name`
Window management	Open / close a window	`Cmd + b + c / Cmd + b + x`
	Move to n^{th} window	`Ctrl + b + n`

Mastering editors

❏ **Vim** – Vim is a popular terminal editor enabling quick and easy file editing, which is particularly useful when connected to a server. The main commands to have in mind are summarized in the table below:

Category	Action	Command
File handling	Go to beginning / end of line	`0 / $`
	Go to first / last line / i^{th} line	`gg / G / i G`
	Go to previous / next word	`b / w`
	Exit file with / without saving changes	`:wq / :q!`
Text editing	Copy line n line(s), where $n \in \mathbb{N}$	`nyy`
	Insert n line(s) previously copied	`p`
Searching	Search for expression containing name__of__pattern	`/name_of_pattern`
	Next / previous occurrence of name__of__pattern	`n / N`
Replacing	Replace old with new expressions with confirmation for each change	`:%s/old/new/gc`

❏ **Jupyter notebook** – Editing code in an interactive way is easily done through jupyter notebooks. The main commands to have in mind are summarized in the table below:

Category	Action	Command
Cell transformation	Transform selected cell to text / code	Click on cell + `m / y`
	Delete selected cell	Click on cell + `dd`
	Add new cell below / above selected cell	Click on cell + `b / a`