

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

Afshine AMIDI and Shervine AMIDI



August 21, 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	<code>git init</code>	
Repository already exists	Copy repository from remote to local	<code>git clone git_address</code>	



**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	<code>git add file</code>	
2. Save snapshot along with descriptive message	<code>git commit -m 'description'</code>	

*Remark 1: `git add .` will have all modified files to the staging area.*

*Remark 2: files that we do not want to track can be listed in the `.gitignore` file.*

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

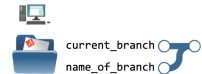
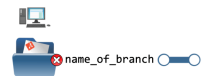
Action	Command	Illustration
Fetch most recent changes from remote branch	<code>git pull name_of_branch</code>	
Push latest local changes to remote branch	<code>git push name_of_branch</code>	

**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:




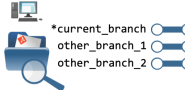
### Bash

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



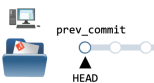
Depending on whether we want to incorporate or discard the branch, we have the following commands:

Action	Command	Illustration
Merge with initial branch	<code>git merge initial_branch</code>	
Remove branch	<code>git branch -D name_of_branch</code>	

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

Action	Command	Illustration
Check status of modified file(s)	<code>git status</code>	
View last commits	<code>git log --oneline</code>	
Compare changes made between two commits	<code>git diff commit_1 commit_2</code>	
View list of local branches	<code>git branch</code>	

❑ **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	<code>git checkout -- file</code>	
Staged	Remove file from staging area	<code>git reset HEAD file</code>	
Committed	Go back to a previous commit	<code>git reset --hard prev_commit</code>	

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

#### Terminal

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

❑ **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	<code>ls path_to_folder   wc -l</code>
Count number of lines in file	<code>cat path_to_file   wc -l</code>
Show last n commands executed	<code>history   tail -n</code>

❑ **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	<code>-name 'certain_name'</code>
	Certain file types (d/f for directory/file)	<code>-type certain_type</code>
	Certain file sizes (c/k/M/G for B/kB/MB/GB)	<code>-size file_size</code>
	Opposite of a given condition	<code>-not [condition]</code>
Actions	Delete selected files	<code>-delete</code>
	Print selected files	<code>-print</code>

*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 of month day 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	<code>tmux / tmux attach</code>
	Leave current session	<code>tmux detach</code>
	List all open sessions	<code>tmux ls</code>
	Remove <code>session_name</code>	<code>tmux kill-session -t session_name</code>
Window management	Open / close a window	<code>Cmd + b + c / Cmd + b + x</code>
	Move to $n^{\text{th}}$ window	<code>Ctrl + b + n</code>

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

□ **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 + <code>m / y</code>
	Delete selected cell	Click on cell + <code>dd</code>
	Add new cell below / above selected cell	Click on cell + <code>b / a</code>