Git Recommended Uses

Practices to get comfortable using and learning Git

Javier Tiá / August 26, 2015

Who am I to talk about Git?

- Using Git professionally since 2011.

Who am I to talk about Git?

- Using Git professionally since 2011.

- Developed tools around Git: repo tool, WSSW Git Hooks and various little utilities.

Who am I to talk about Git?

- Using Git professionally since 2011.

- Developed tools around Git: repo tool, WSSW Git Hooks and various little utilities.

- Migrated severals projects to Git from Clearcase, CVS and Subversion.

Who am I to talk about Git?

- Using Git professionally since 2011.

- Developed tools around Git: repo tool, WSSW Git Hooks and various little utilities.

- Migrated severals projects to Git from Clearcase, CVS and Subversion.

- Solved hundreds of issues helping people using Git. Today I still do it, but no hundreds 😊.

About this presentation

- No a proper Git training. It could take days.

- It's a collection of recommendations have been put together through some years of experience.

- From this presentation you can take some knowledge and improve your frequent Git routine.

Why 'Git Recommended Uses' presentation?

1. People struggle while are learning Git.

Why 'Git Recommended Uses' presentation?

1. People struggle while are learning Git.

2. Git is a powerful tool, but it has some hard hills to climb.

Why 'Git Recommended Uses' presentation?

1. People struggle while are learning Git.

2. Git is a powerful tool, but it has some hard hills to climb.

3. At the beginning poor documentation and severals ways to get same result.

Why 'Git Recommended Uses' presentation?

1. People struggle while are learning Git.

2. Git is a powerful tool, but it has some hard hills to climb.

3. At the beginning poor documentation and severals ways to get same result.

4. Git is not Subversion.

Why Git and no another SCM

Job Trends from Indeed.com
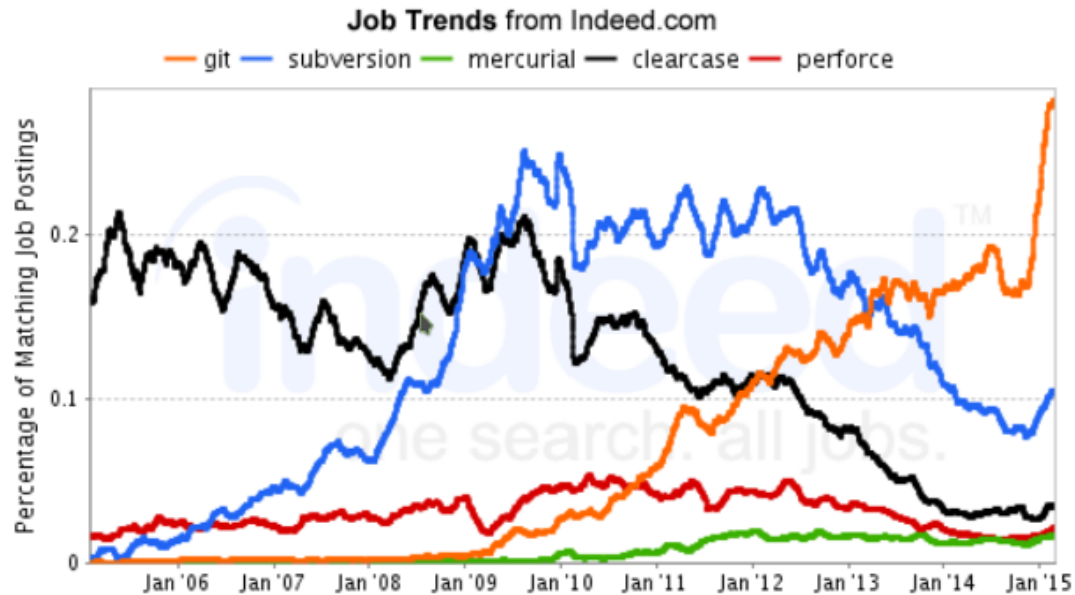
How to self-study Git?

- Mindset: Think Git as you are exploring a tree.

How to self-study Git?

- Mindset: Think Git as you are exploring a tree.

- Get started with:

    - http://try.github.io/

    - http://gitimmersion.com/

    - http://git-scm.com/book/

How to self-study Git?

- Mindset: Think Git as you are exploring a tree.

- Get started with:

  - http://try.github.io/

  - http://gitimmersion.com/

  - http://git-scm.com/book/

- Complement always with:

  - `git help <COMMAND> || git help --all || git help --guides`

How to self-study Git?

- Mindset: Think Git as you are exploring a tree.

- Get started with:

  - http://try.github.io/

  - http://gitimmersion.com/

  - http://git-scm.com/book/

- Complement always with:

  - `git help <COMMAND> || git help --all || git help --guides`

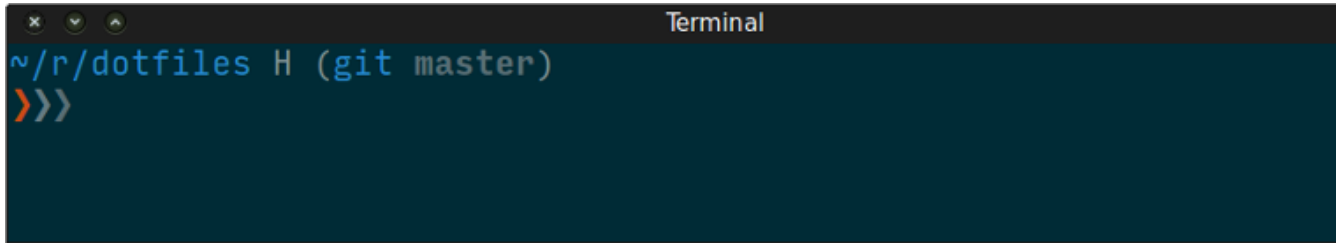 - Making questions with a Web searcher.

How to self-study Git?

- Mindset: Think Git as you are exploring a tree.
- Get started with:
    - http://try.github.io/
    - http://gitimmersion.com/
    - http://git-scm.com/book/
- Complement always with:
    - `git help <COMMAND> || git help --all || git help --guides`
    - Making questions with a Web searcher.
    - Practice in a terminal.

Use latest stable Git version

Assuming Ubuntu as Distribution.

```
$ sudo add-apt-repository ppa:git-core/ppa

$ sudo apt-get update

$ sudo apt-get install git
```

Get a better CLI prompt


```
Terminal
~/r/dotfiles H (git master)
>>>
```

https://github.com/nojhan/liquidprompt

Identify yourself

Remember now you are an author. Identify yourself:

```
$ git config --global user.name "__YOUR NAME__"

$ git config --global user.email "__YOUR EMAIL ADDRESS__"
```

Increase Git usability by leveraging *.gitconfig*

Set per repository or global options.

Use `git config` to Get, Set and Unset options.

Don't forget look for more details: `git help config`.

Common Git aliases

```
$ git config --global alias.br branch

$ git config --global alias.ci commit

$ git config --global alias.co checkout

$ git config --global alias.st status

$ git config --global alias.ls ls-files

$ git config --global alias.rb rebase

$ git config --global alias.mg merge
```

Use Interactive mode to commit

Interactively choose hunks between the index and the work tree and add
them to the index:

```
$ git add --patch
```

Use Interactive mode to commit

Interactively choose hunks between the index and the work tree and add them to the index:

```
$ git add --patch
```

It gives the user a chance to review the changes before commit them.

Use Interactive mode to commit

Interactively choose hunks between the index and the work tree and add
them to the index:

```
$ git add --patch
```

It gives the user a chance to review the changes before commit them.

Consider use a shell alias:

```
$ alias giA='git add --patch'
```

How to write a commit message

Structure of a commit message:

> Subject line
> *BLANK LINE*
> Body message

How to write a commit message

Structure of a commit message:

> Subject line
> *BLANK LINE*
> Body message

Seven Rules:

1. Write subject line thinking: What am I doing?

How to write a commit message

Structure of a commit message:

> Subject line
> *BLANK LINE*
> Body message

Seven Rules:

1. Write subject line thinking: What am I doing?

2. Use imperative mood in subject line: no -ing, -ed.

How to write a commit message

Structure of a commit message:

> Subject line
> *BLANK LINE*
> Body message

Seven Rules:

1. Write subject line thinking: What am I doing?

2. Use imperative mood in subject line: no -ing, -ed.

3. Do not end subject line with a period.

How to write a commit message

Structure of a commit message:

> Subject line
> *BLANK LINE*
> Body message

Seven Rules:

1. Write subject line thinking: What am I doing?

2. Use imperative mood in subject line: no -ing, -ed.

3. Do not end subject line with a period.

4. Limit subject line between 50 to 70 characters.

How to write a commit message

Structure of a commit message:

> Subject line
> *BLANK LINE*
> Body message

Seven Rules:

1. Write subject line thinking: What am I doing?

2. Use imperative mood in subject line: no -ing, -ed.

3. Do not end subject line with a period.

4. Limit subject line between 50 to 70 characters.

5. Insert a blank line between subject line and body message.

How to write a commit message

Structure of a commit message:

> Subject line
> *BLANK LINE*
> Body message

Seven Rules:

1. Write subject line thinking: What am I doing?

2. Use imperative mood in subject line: no -ing, -ed.

3. Do not end subject line with a period.

4. Limit subject line between 50 to 70 characters.

5. Insert a blank line between subject line and body message.

6. Write body message thinking: Why am I doing it? vs How am I doing it?

How to write a commit message

Structure of a commit message:

> Subject line
> *BLANK LINE*
> Body message

Seven Rules:

1. Write subject line thinking: What am I doing?

2. Use imperative mood in subject line: no -ing, -ed.

3. Do not end subject line with a period.

4. Limit subject line between 50 to 70 characters.

5. Insert a blank line between subject line and body message.

6. Write body message thinking: Why am I doing it? vs How am I doing it?

7. Limit body message to 72 characters.

Fit Commit Tool

A Git hook to validate your commit messages based on community standards.

```
$ git commit
Adding a cool feature
foobar foobar foobar,
foobar foobar foobar foobar foobar foobar foobar foobar foobar foobar foobar

1: Error: Message must use imperative present tense.
2: Error: Second line must be blank.
3: Error: Lines should be <= 72 chars. (76)

Force commit? [y/n/e] █
```

https://github.com/m1foley/fit-commit

Merge vs Rebase

One of most controversial Git discussions, just search: Merge vs Rebase.

Merge vs Rebase

One of most controversial Git discussions, just search: Merge vs Rebase.

Know when use *merge* or *rebase* add significant added value. It produces a clean and semantically correct history graph.

Merge vs Rebase

One of most controversial Git discussions, just search: Merge vs Rebase.

Know when use *merge* or *rebase* add significant added value. It produces a clean and semantically correct history graph.

Three Rules:

1. Work always with topic or temporal branches, avoid work over master or features branches.

Merge vs Rebase

One of most controversial Git discussions, just search: Merge vs Rebase.

Know when use *merge* or *rebase* add significant added value. It produces a clean and semantically correct history graph.

Three Rules:

1. Work always with topic or temporal branches, avoid work over master or features branches.

2. Use `git merge` when incorporating an entire feature set into another branch, use `git rebase` for rest of cases.

Merge vs Rebase

One of most controversial Git discussions, just search: Merge vs Rebase.

Know when use *merge* or *rebase* add significant added value. It produces a clean and semantically correct history graph.

Three Rules:

1. Work always with topic or temporal branches, avoid work over master or features branches.

2. Use `git merge` when incorporating an entire feature set into another branch, use `git rebase` for rest of cases.

3. Golden rule: Never use `git rebase` over published changes, only over new and no pushed changes.

Keep clean and correct Git history

* Always rebase your local commits and preserve all merges:

```
$ git config --global pull.rebase preserve
```

   Require Git v1.8.5+

Keep clean and correct Git history

* Always rebase your local commits and preserve all merges:

```
$ git config --global pull.rebase preserve
```

Require Git v1.8.5+

* For merging features and avoid lose the branch information of which a commit was originally made:

git	git config (Permanent)	git config (Just to master)
merge	merge.ff	branch.master.mergeoptions
--no-ff FEATURE	false	--no-ff

Keep clean and correct Git history

* Always rebase your local commits and preserve all merges:

```
$ git config --global pull.rebase preserve
```

   Require Git v1.8.5+

* For merging features and avoid lose the branch information of which a commit was originally made:

git	git config (Permanent)	git config (Just to master)
merge	merge.ff	branch.master.mergeoptions
--no-ff FEATURE	false	--no-ff

* For merging fixes use `--ff-only` to fast-forward:

git	git config (Permanent)	git config (Just to master)
merge	merge.ff	branch.master.mergeoptions
--ff-only FIX	only	--ff-only

Rebase interactive

It allows to navigate for each commit deciding interactively what to do.

```
$ git rebase -i
```

Rebase interactive

It allows to navigate for each commit deciding interactively what to do.

```
$ git rebase -i
```

You can pick (or delete), reword commit message, edit code, squash, fixup or exec an command.

Rebase interactive

It allows to navigate for each commit deciding interactively what to do.

```
$ git rebase -i
```

You can pick (or delete), reword commit message, edit code, squash, fixup or exec an command.

Automatically modify the rebase todo list:

```
$ git config --global rebase.autoSquash true
```

Rebase interactive

It allows to navigate for each commit deciding interactively what to do.

```
$ git rebase -i
```

You can pick (or delete), reword commit message, edit code, squash, fixup or exec an command.

Automatically modify the rebase todo list:

```
$ git config --global rebase.autoSquash true
```

Automatically stash local changes, apply it when rebase finish:

```
$ git config --global rebase.autoStash true
```

Require: Git v2.6.x+

Rebase concept

Resolving conflicts

`--ours` option: current branch changes checked out.

`--theirs` option: changes coming in via merge or rebase.

Resolving conflicts

--ours option: current branch changes checked out.

--theirs option: changes coming in via merge or rebase.

Normal way:

```
$ git checkout --ours/--theirs <FILE>

$ git add <FILE>
```

Resolving conflicts

`--ours` option: current branch changes checked out.

`--theirs` option: changes coming in via merge or rebase.

Normal way:

```
$ git checkout --ours/--theirs <FILE>

$ git add <FILE>
```

Using alias:

```
$ git config --global alias.our = ours = "!f() { \
      git checkout --ours $@ && git add $@; }; f"

$ git ours <FILE>
```

Resolving conflicts

`--ours` option: current branch changes checked out.

`--theirs` option: changes coming in via merge or rebase.

Normal way:

```
$ git checkout --ours/--theirs <FILE>

$ git add <FILE>
```

Using alias:

```
$ git config --global alias.our = ours = "!f() { \
      git checkout --ours $@ && git add $@; }; f"

$ git ours <FILE>
```

For repeated resolution of conflicts:

```
$ git config --global rerere.enabled true
```

Switch context at will

Saves local modifications and produce a clean state:

```
$ git stash
```

Switch context at will

Saves local modifications and produce a clean state:

```
$ git stash
```

Allows to switch context to solve issues like: fix a local commit, checkout another branch and more.

Switch context at will

Saves local modifications and produce a clean state:

```
$ git stash
```

Allows to switch context to solve issues like: fix a local commit, checkout another branch and more.

Prevent unwanted deletions from `git reset --hard/--merge`.

Switch context at will

Saves local modifications and produce a clean state:

```
$ git stash
```

Allows to switch context to solve issues like: fix a local commit, checkout another branch and more.

Prevent unwanted deletions from `git reset --hard/--merge`.

If stashed changes will live a long time, name it:

```
$ git stash save 'NAME MESSAGE'
```

Switch context at will

Saves local modifications and produce a clean state:

```
$ git stash
```

Allows to switch context to solve issues like: fix a local commit, checkout another branch and more.

Prevent unwanted deletions from `git reset --hard/--merge`.

If stashed changes will live a long time, name it:

```
$ git stash save 'NAME MESSAGE'
```

Don't forget to look in `git help stash` for further details.

Git clean

Remove recursively files that are not under version control. Files like:
generated by build system.

Git clean

Remove recursively files that are not under version control. Files like: generated by build system.

Prevent unwanted deletion by running first:

```
$ git clean -n
```

It shows all files would be removed.

Git clean

Remove recursively files that are not under version control. Files like: generated by build system.

Prevent unwanted deletion by running first:

```
$ git clean -n
```

It shows all files would be removed.

Look in `git help clean` for further details.

Useful little recommendations

Add color helpers to read text easier and see what Git is doing:

```
$ git config --global color.ui true
```

Useful little recommendations

Add color helpers to read text easier and see what Git is doing:

```
$ git config --global color.ui true
```

Copy/move a file in an one commit. Git doesn't difference between removing files and changes in it.

xkcd: Git

References

- This presentation: https://github.com/jetm/git-recommended-uses/ or pdf file.

- Presentation made with: http://remarkjs.com/

- My .gitconfig file

Contact information

- javier.tia@gmail.com / javier.tia@hpe.com

Questions?

Backup slides

Little Git History

- Created in 2005 by Linus Torvalds and maintained today by Junio Hamano.

Little Git History

- Created in 2005 by Linus Torvalds and maintained today by Junio Hamano.

- Meaning 'git' as word: unpleasant person.

Little Git History

- Created in 2005 by Linus Torvalds and maintained today by Junio Hamano.

- Meaning 'git' as word: unpleasant person.

- Git arrived as a necessity.

Git Reset/Checkout
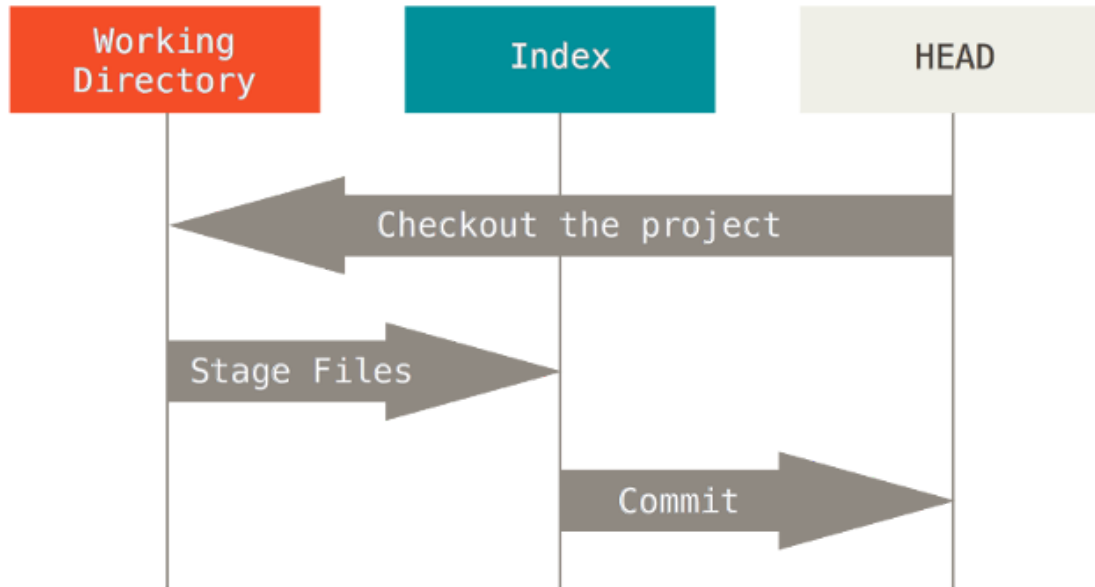
Protect bare/server repositories

No rewriting history:

```
$ git config receive.denyNonFastForwards true
```

No deleting history:

```
$ git config receive.denyDeletes true
```

Check object consistency:

```
$ git config receive.fsckObjects true
```