GitHub good practices

Table of Contents

1. Correspondences Scrum/GitHub/Gitlab

Table 1. Correspondences between Artefacts

Scrum	GitHub	GitLab
User Story	Issues	Issues
Task	Task Lists / dependency issues	Task Lists
Epic	??	Epics
Points/Estimation	Weights ($)	Weights
Product Backlog	Issues Lists	Issues Lists
Priorities	Labels	Labels
Sprint	Milestone	Milestone
Burdown Chart	??	Burdown Chart
Agile board	Project board	Issue Board

2. Build

Let's do a quick poll about build tools

[sondage] | *sondage.png*

Figure 1. 2021 promotion build tool usage

2014 Study:

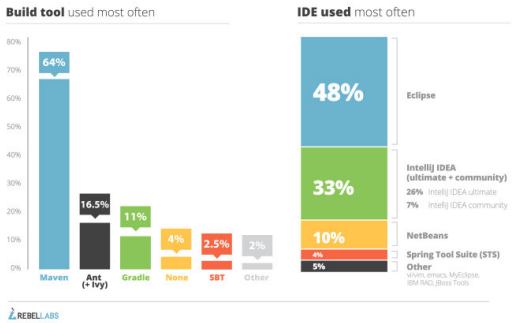

Figure 2. Ant vs Maven vs Gradle (source here)

Ant example (cf. source)

```
<project xmlns:ivy="antlib:org.apache.ivy.ant" name="java-build-tools" default="jar">

    <property name="src.dir" value="src"/>
    ...
    <path id="lib.path.id">
        <fileset dir="${lib.dir}" />
    </path>

    <target name="clean">
        <delete dir="${build.dir}"/>
    </target>

    <target name="compile">
        <mkdir dir="${classes.dir}"/>
        <javac srcdir="${src.dir}" destdir="${classes.dir}"
classpathref="lib.path.id"/>
    </target>

    <target name="jar" depends="compile">
        <mkdir dir="${jar.dir}"/>
        <jar destfile="${jar.dir}/${ant.project.name}.jar" basedir="${classes.dir}"/>
    </target>

</project>
```

Maven example (cf. source)

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <groupId>com.technologyconversations</groupId>
    <artifactId>java-build-tools</artifactId>
    <packaging>jar</packaging>
    <version>1.0</version>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.11</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.3.2</version>
            </plugin>
        </plugins>
    </build>

</project>
```

Gradle example (cf. source)

```
apply plugin: 'java'
apply plugin: 'checkstyle'
apply plugin: 'findbugs'
apply plugin: 'pmd'

version = '1.0'

repositories {
    mavenCentral()
}

dependencies {
    testCompile group: 'junit', name: 'junit', version: '4.11'
    testCompile group: 'org.hamcrest', name: 'hamcrest-all', version: '1.3'
}
```

3. Automate issue branches

https://github.com/marketplace/actions/create-issue-branch

Add this to your workflow YAML configuration:

```
on:
    issues:
        types: [assigned]
    issue_comment:
        types: [created]
    pull_request:
        types: [closed]

jobs:
    create_issue_branch_job:
        runs-on: ubuntu-latest
        steps:
        - name: Create Issue Branch
          uses: robvanderleek/create-issue-branch@master
          env:
            GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

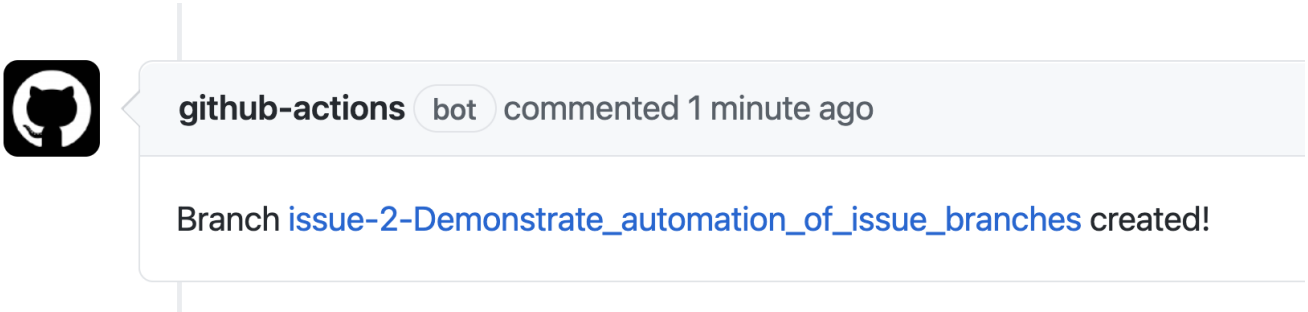

Figure 3. As soon as the issue is assigned...

4. Use tags

```
git tag 1.1.0 -m "Release 1.1.0"
git push origin tag 1.1.0
```

Tagging practices:

- You don't tag branches. You tag commits!
- You should add a tag to mark a released version. If you then need to make bug fixes to that release you would create a branch at the tag
- If you checkout a tag, you will need to create a branch to start working from it

More here.

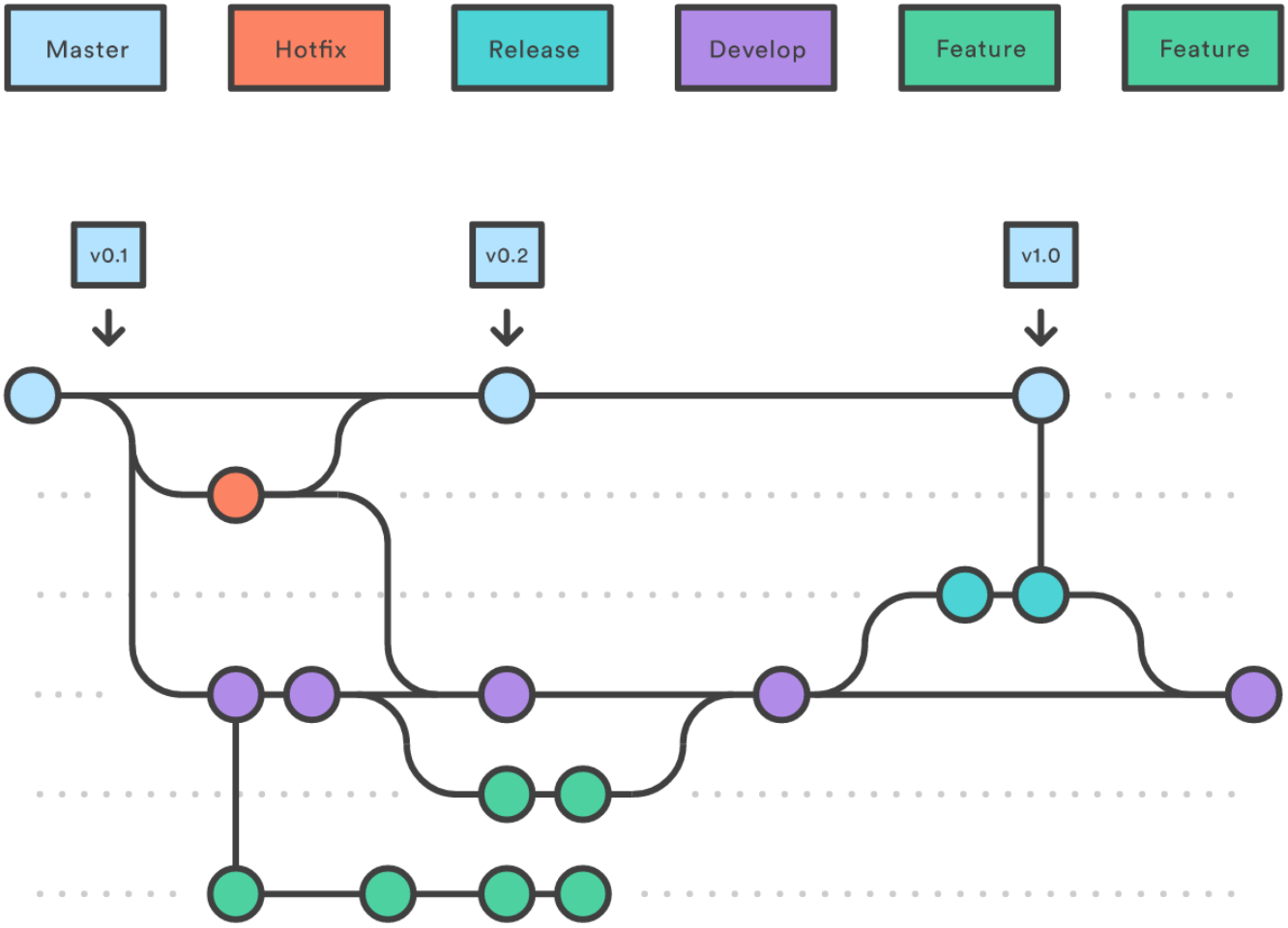

Figure 4. Tags in git flow (source here)

5. Meaningful `.gitignore` file

This repo `.gitignore` file

```
# Output directory for HTML files
output/
Gemfile.lock
*.html
.DS_Store


topics
```

https://gitignore.io

Example for `Node.js` *(only beginning!)*

```
# Created by https://www.toptal.com/developers/gitignore/api/node
# Edit at https://www.toptal.com/developers/gitignore?templates=node

### Node ###
# Logs
logs
*.log
npm-debug.log*
yarn-debug.log*
yarn-error.log*
lerna-debug.log*
```

6. Meaningful commit messages

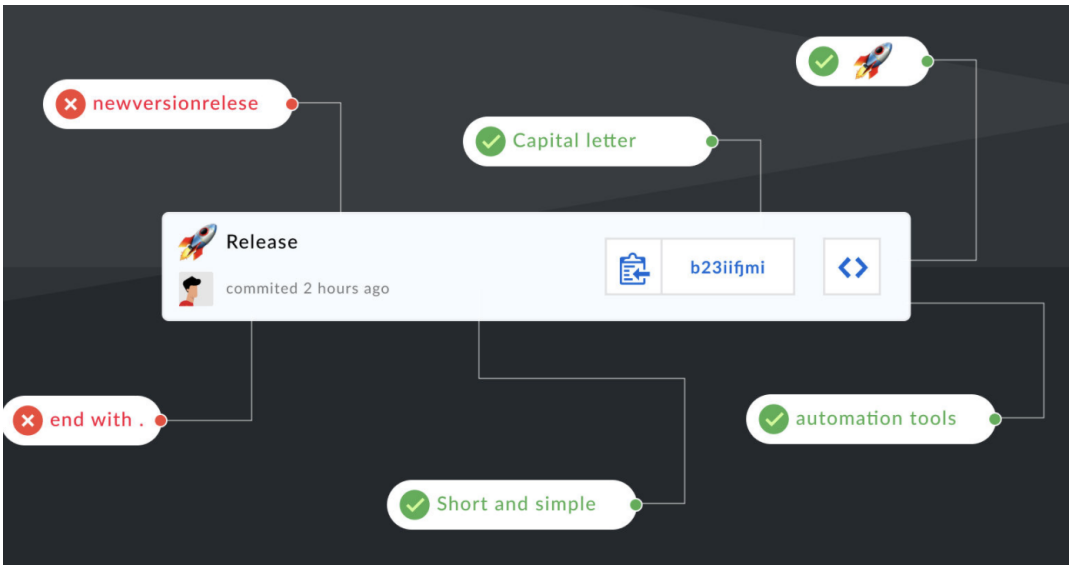

Figure 5. Example of conventions (source here)

Example of `~/.gitconfig` file

```
# Git Commit, Add all and Push — in one step.
cap = "!f() { git add .; git commit -m \"$@\"; git push; }; f"

# NEW.
new = "!f() { git cap \"🎉 NEW: $@\"; }; f"
# IMPROVE.
imp = "!f() { git cap \"🔧 IMPROVE: $@\"; }; f"
# FIX.
fix = "!f() { git cap \"🔨 FIX: $@\"; }; f"
# RELEASE.
rlz = "!f() { git cap \"🚀 RELEASE: $@\"; }; f"
# DOC.
doc = "!f() { git cap \"📝 DOC: $@\"; }; f"
# TEST.
tst = "!f() { git cap \"🚨 TEST: $@\"; }; f"
```

Emoji	Description
🎉 `:tada:`	When you added a cool new feature.
🔧 `:wrench:`	When you refactored / improved a small piece of code.
🔨 `:hammer:`	When you refactored / improved large parts of the code.
✨ `:sparkles:`	When you applied clang-format.
🎨 `:art:`	When you improved / added assets like themes.
🚀 `:rocket:`	When you improved performance.
📝 `:memo:`	When you wrote documentation.
🐞 `:beetle:`	When you fixed a bug.
🔀 `:twisted_rightwards_arrows:`	When you merged a branch.
🔥 `:fire:`	When you removed something.
🚚 `:truck:`	When you moved / renamed something.

Figure 6. Example of emoji usage convention

7. Useful links

- https://www.datree.io/resources/github-best-practices