GIT Ticket Workflow

Version 47 Updated by Robin C. on 20 Aug 2015. Previous Versio

Check out development branch

To check out our development branch please:

1. Clone our GitHub repo

```
git clone git@github.com:VHAINNOVATIONS/Mental-Health-eScreening.git
```

2. Check out the po_ready branch

```
git checkout -b po_ready origin/po_ready
```

Ticket Workflow

The ticket workflow decided upon by the team is as follows:

1. When a developer is assigned a ticket, he/she becomes the owner of that ticket and will do the following:
    1. Set the ticket's status to Accepted when work is to be started
    2. Checkout a new ticket branch from the po_ready branch using the naming convention of: a letter "t" followed by the ticket number (e.g. t565).

```
git fetch
git checkout -b t565 origin/po_ready
```

    3. Push the branch to GitHub to backup work and/or if other developers must also contribute changes.

```
git push -u origin t565
```

    4. Commit changes to the ticket's branch as needed

```
git status
git add <some-file>
git commit -m "#565 added feature..."
```

2. When the ticket owner has completed all work for a ticket he/she merges the ticket's branch into our *resolved* branch for testing:
    1. Make sure the ticket's branch is checked out

```
git checkout t565
```

    2. Pull the latest changes from GitHub

```
git pull
```

    3. Push the ticket's final changes to GitHub

```
git push
```

    4. Checkout the *resolved* branch

```
git checkout resolved
```

    5. Merge the ticket's branch into the *resolved* branch

```
git merge t565
```

    6. Push the merged *resolved* branch

```
git push
```

3. If the QA team rejects a ticket as not done the developer will make changes to the ticket's branch and merge those new changes into the *resolved* branch.
4. When the QA team accepts that a ticket is defect free, the ticket owner will merge the ticket's branch into po_ready:

```
git checkout t565
```

2. Pull the latest changes from GitHub

```
git pull
```

3. Push the ticket's final changes to GitHub

```
git push
```

4. Checkout the po_ready branch

```
git checkout po_ready
```

5. Merge the ticket's branch into the po_ready branch

```
git merge t565
```

6. Push the merged *resolved* branch

```
git push
```

5. If the PO rejects a ticket as not done then do the following:
    1. developer goes back to the ticket's branch and makes changes to it
    2. **These new changes need to be first merged to the resolved branch**
    3. The QA team tests the new changes using the Test sandbox instance
    4. If the ticket passes all QA tests, then the ticket's branch is merged into po_ready branch again
    5. The demo instance is then rebuilt and the PO can retest
6. When the sprint has completed and all tickets with a state of "Verified & Ready For PO Acceptance" have been closed, the following action will be carried out:
    1. An audit of all ticket branches for the current sprint is conducted to ensure all ticket branches are found in both *resolved* and po_ready
    2. Ming will merge the po_ready branch into the master
    3. Ming will tag master with the sprint number (e.g. "sprint_22")

```
git tag -a sprint22 -m "Sprint 22 release" master
git push origin sprint22
```

    4. Chuck will update the integration (int) sandbox instance
    5. Ming will update the San Diego test server with the master branch code
    6. All ticket branches that were merged into Master should be deleted.
    7. The following git command is to be issued by each individual developer to delete their local ticket branch once the ticket is closed

```
git branch -d t565
```

    8. The following git command is to be run by Ming only as this will remove the branch from GitHub

```
git push origin :t565
```

Sandbox Instances

The following server instances are build from the cooresponding branch:

- eScreening Test is built from the *resolved* branch
- eScreening Demo is built from the po_ready branch
- eScreening Int (integration) is built from the master branch

Other Workflow Paths

The above list of steps describes the happy path. Below we describe other possible paths that may arise.

Checking out a remote branch

If you are helping the ticket owner and need to checkout a remote branch that has already been created please follow these steps:

1. Update your local with latest list of remote branches

2. Check out the remote branch

```
git checkout t565
```

Helpful Commands

Git Configuration

Git is deep and very interesting. You can do so much with Git. To understand Git. Please take time and listen to this tutorial by Andrew Burgess at tuts.plus.com https://code.tutsplus.com/courses/git-essentials

PS: You will need to buy a subscription of tutsplus.com. Worth every penny.

when you want to apply same comment to some forgotten files

```
git commit --amend --no-edit
```

when you want to know which files are committed locally but due to be pushed. **This happens to me all the time**

```
git diff --numstat origin/resolved
```

when you do not remember if you have pushed your changes and want to find which comments are committed locally but pending push

```
git log origin/resolved..HEAD
```

suppose you are on a branch t646 and because you only want to borrow pom.xml from resolved branch. How can we do this using git.

```
git checkout resolved -- pom.xml
Here 'resolved' is the branch you want to borrow file from and pom.xml is the relative path of 'pom.xml.'
Suppose the file was some an sql file then you would have used src/main/sql/sql_file.sql
```

to get a list of all branches which have been merged into the resolved branch:

```
git branch -a --merged resolved
```

to rename a ticket's branch:

```
1. Switch to the branch you want to rename, then rename it it locally:
  git checkout old_branch
  git branch -m old_branch new_branch
2. Remove old branch from remote repository:
  git push origin :old_branch
3. Push the renamed local branch to remote repository:
  git push origin new_branch
```

to see files recently pushed

```
git log --name-status HEAD^..HEAD
```

when you have to overwrite only one file:

```
git fetch
git checkout origin/resolved <filepath>
```

when you have to overwrite all changed files:

```
git fetch
git reset --hard origin/resolved
```

to see a list of files to be pushed

```
git diff --stat origin/resolved
```

when you want to view unpublished GIT files

when you have to revert some file already committed

```
git checkout resolved pom.xml
git reset HEAD pom.xml
git checkout -- pom.xml
```

Before you push to origin, if you want to undo the last commit (to change something)

```
git reset --soft HEAD~1
<< edit files as necessary >>
git add ....
git commit
```

when you want to compare the files changed between ticket branch and merge target:

```
git checkout t656
git pull origin resolved
git diff --name-only resolved t656
```

when you want to overwrite local change with another branch

```
git checkout origin/resolved -- pom.xml
```

to get a better log of changes to a file even including changes introduced during merges:

```
git log -m --no-max-parents -p /path/to/file
```

if you are in the middle of a merge and you find out that you merged a file incorrectly and need to do it again. You can reset the file to how it was before your merge with this:

```
git checkout -m /path/to/file
```

To get a list of all files changed since a tag:

```
git diff --name-status <tag_name>
```

Helpful tools

To better track each ticket's branch and whether it was merged into resolved and/or master, use auditBranches.sh
It also shows any commits that are found in resolved that are not in master and vice versa (i.e. set difference).
You have to be in the eScreening directory managed by git when you run it and a list of ticket number can be given.
Scala has to be installed.
It assumes that the branch name has the ticket number in it.